DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9644

RESOURCE-CONSTRAINED PROJECT SCHEDULING

A survey of recent developments

by

W. Herroelen

E. Demeulemeester

B. De Reyck

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9644

RESOURCE-CONSTRAINED PROJECT SCHEDULING

A survey of recent developments

by

W. Herroelen

E. Demeulemeester

B. De Reyck

D/1996/2376/44

RESOURCE-CONSTRAINED PROJECT SCHEDULING
A survey of recent developments

Willy HERROELEN • Erik DEMEULEMEESTER • Bert DE REYCK

Katholieke Universiteit Leuven
Department of Applied Economics
Naamsestraat 69, B-3000 Leuven (Belgium)
Tel 32-16-32 69 70 (32 69 72 or 32 69 66)
Fax 32-16-32 67 32
e-mail: willy.herroelen, erik.demeulemeester or bert.dereyck@econ.kuleuven.ac.be

August 1996

Background text for the invited address delivered by W. Herroelen at the semi-plenary session of the Scheduling Section of the *Symposium on Operations Research 1996*, Annual Conference of the Deutsche Gesellschaft für Operations Research (DGOR) and the Gesellschaft für Mathematik, Ökonomie und Operations Research (GMÖOR), co-sponsored by the International Federation for Information Processing, Working Group WG 7.4 Discrete Optimization, Technical University Braunschweig, Germany, September 4-6, 1996.

This research was supported by N.F.W.O. contract G.0131.96, which is gratefully acknowledged.

RESOURCE-CONSTRAINED PROJECT SCHEDULING
A survey of recent developments

Willy HERROELEN • Erik DEMEULEMEESTER• Bert DE REYCK
Katholieke Universiteit Leuven, Department of Applied Economics
Naamsestraat 69, B-3000 Leuven (Belgium)

ABSTRACT

Resource-constrained project scheduling involves the scheduling of project activities subject to precedence and resource constraints in order to meet the objective(s) in the best possible way. The area covers a wide variety of problem types. The objective of this paper is to provide a survey of what we believe are important recent developments in the area. Our main focus will be on the recent progress made in and the encouraging computational experience gained with the use of *optimal* solution procedures for the basic resource-constrained project scheduling problem (RCPSP) and important extensions.

The RCPSP involves the scheduling of a project to minimize its duration subject to zero-lag finish-start precedence constraints of the PERT/CPM type and constant availability constraints on the required set of renewable resources. We discuss recent striking advances in dealing with this problem using a new depth-first branch-and-bound procedure, elaborating on the effective and efficient branching scheme, bounding calculations and dominance rules, and discuss the potential of using truncated branch-and-bound. We derive a set of conclusions from the research on optimal solution procedures for the basic RCPSP and subsequently illustrate how effective and efficient branching rules and several of the strong dominance and bounding arguments can be extended to a rich and realistic variety of related problems. The preemptive resource-constrained project scheduling problem (PRCPSP) relaxes the nonpreemption condition of the RCPSP, thus allowing activities to be interrupted at integer points in time and resumed later without additional penalty cost. The generalized resource-constrained project scheduling problem (GRCPSP) extends the RCPSP to the case of precedence diagramming type of precedence constraints (minimal finish-start, start-start, start-finish, finish-finish precedence relations), activity ready times, deadlines and variable resource availabilities. The resource-constrained project scheduling problem with generalized precedence relations (RCPSP-GPR) allows for start-start, finish-start, start-finish and finish-finish constraints with minimal *and* maximal time lags. The MAX-NPV problem aims at scheduling project activities in order to maximize the net present value of the project in the absence of resource constraints. The resource-constrained project scheduling problem with discounted cash flows (RCPSP-DC) aims at the same non-regular objective in the presence of resource constraints. The resource availability cost problem (RACP) aims at determining the cheapest resource availability amounts for which a feasible solution exists that does not violate the project deadline. In the discrete time/cost trade-off problem (DTCTP) the duration of an activity is a discrete, non-increasing function of the amount of a single *nonrenewable* resource committed to it. In the discrete time/resource trade-off problem (DTRTP) the duration of an activity is a discrete, non-increasing function of the amount of a single *renewable* resource. Each activity must then be scheduled in one of its possible execution modes. In addition to time/resource trade-offs, the multi-mode project

scheduling problem (MRCPSP) allows for resource/resource trade-offs and constraints on renewable, nonrenewable and doubly-constrained resources. We report on recent computational results and end with overall conclusions and suggestions for future research.

Keywords: Project scheduling; Resource constraints; Branch-and-bound.

1. Introduction

Scheduling and sequencing is concerned with the optimal allocation of scarce resources over time. *Scheduling* deals with defining which activities are to be performed at a particular time. *Sequencing* concerns the ordering in which the activities have to be performed. The allocation of scarce resources over time has been the subject of extensive research since the early days of operations research in the mid 1950s. The result is a vast and difficult to digest literature and a considerable gap between scheduling theory and shop floor practice. Practitioners blame scheduling theoreticians to spend scarce research money for studying toy problems such as sequencing a set of simultaneously available unordered jobs with known durations on a never failing machine in order to optimize irrelevant objective functions. Theoreticians blame practitioners for their ignorance about the recent developments, their reluctance in applying useful theory, or their over-enthusiasm in applying scheduling procedures miles away from their natural field of application. Despite this mutual 'interest', major issues largely remain unresolved in practice, and scheduling and sequencing problems remain the subject of intensive research.

All this does not come by surprise. Scheduling and sequencing theory, more than any other field in the area of operations management and operations research, is characterized by a virtually unlimited number of *problem types*. The terminology arose in the processing and manufacturing industries and most research has traditionally been focused on ***deterministic machine scheduling*** (see the books by Ashour (1972), Baker (1974), Bellmann et al. (1982), Blazewicz et al. (1993), Brucker (1995), Conway et al. (1967), French (1982), Herroelen (1991), Morton and Pentico (1993), Pinedo (1995), Rinnooy Kan (1976) and Tanaev et al. (1994a,b). In this context the type of resource is traditionally considered to be a *machine* that can perform at most one activity at a time.

The activities are commonly referred to as *jobs*, and it is usually assumed that a job is processed by at most one machine at a time. The processing of a job on a machine is called an *operation*. The machine environment is quite diverse. In a *single machine* environment, each job has only one operation (one-phase production). In a *parallel machine* environment each job also requires just one operation, but that operation may be performed on any of the machines. When the machines are identical, the processing time of a job is the same on all machines. When the machines are uniform, the processing time varies as a function of a given reference speed. When the machines are unrelated, the processing time of a job again varies, but now in a completely arbitrary fashion.

In multistage production, a job consists of a number of operations. *Technological precedence constraints* demand that each job should be processed through the machines in a particular order. For general *job-shop* problems there are no restrictions upon the form of the technological constraints. When all the jobs share the same processing order we have a *flow-shop* problem. In the special case of an *open shop*, each job has to be processed on each machine, but there is no particular order to follow. In open shops, the schedule determines not only the order in which machines process the jobs, but also that in which the jobs pass between machines. Jobs are characterized by a *ready time* (*release date*) which denotes the time at which the job becomes available for processing. The time by which the job should be finished is called the *due date* (if the due date may not be exceeded, the term *deadline* is

often used). It is possible to consider situations were jobs may be split or not. Each operation takes a certain length of time, the *processing time*, to be performed. In addition, operations may be subject to sequence dependent *set up times*.

The *performance criteria* are numerous: minimize schedule length (makespan); minimize mean (weighted) flow time; minimize mean or maximum lateness or tardiness (lateness is the difference between a job's completion time and its due date - the lateness for an early job being negative; when a job is completed after its due date, it is tardy - tardiness being the maximum of zero and the lateness); minimize the number of tardy jobs; maximize throughput (number of jobs completed per time unit), etc.. Sometimes combined scheduling criteria are used: minimize mean flow time subject to no jobs late, search for the shortest mean flow time schedule, search for a schedule in which no job is early nor tardy (just-in-time), etc.. Problems can be studied in a *static* environment (all jobs simultaneously available) or in a *dynamic* environment (jobs have unequal ready times). Problems may be considered to be *deterministic* or *stochastic*.

Over the years, several (irrealistic) assumptions of the basic machine scheduling problems have been relaxed. A natural extension involves the presence of *additional resources*, where each resource has a limited size and each job requires the use of a part of each resource during its execution (Gargeya and Deane 1996). This leads us to the area of ***resource-constrained project scheduling*** which involves the scheduling of project activities subject to precedence and resource constraints. The field covers again a tremendous variety of problem types. Certain types of resources are depleted by use (e.g. nonrenewable resources such as money and energy). Resources may be available in an amount that varies over time in a predictable manner (e.g. seasonal labor) or in an unpredictable manner (e.g. equipment vulnerable to failure). Resources may be shared among several jobs, and a job may need several resources. The resource amounts required by a job may vary during its processing, and the processing time itself could depend on the amount or type of resource allocated, as in the case of the above mentioned uniform or unrelated parallel machines. Over the past few years, extensive research efforts have resulted in new results on the level of problem classification and complexity, and new optimal and suboptimal solution approaches.

This article aims at providing a guided tour through what we believe to be important recent developments in the area of deterministic resource-constrained project scheduling. Our main focus will be on the recent progress made with optimal branch-and-bound procedures for the basic resource-constrained project scheduling problem (RCPSP) and fundamental extensions of successful branching schemes, dominance and bounding arguments to important related problem types.

The organization of this paper is as follows. §2 focuses on the classical resource-constrained project scheduling problem (RCPSP). §3 deals with the preemptive resource-constrained project scheduling problem (PRCPSP). §4 concentrates on the recent developments of models dealing with generalized precedence relations. §5 addresses the problem of maximizing the net present value of projects, as an illustration of the use of non-regular measures of performance. §6 discusses the problem of minimizing resource availability costs. §7 is devoted to project scheduling problems with time/resource and resource/resource trade-offs. §8 is reserved for our overall conclusions.

2. The resource-constrained project scheduling problem (RCPSP)

We assume that a project is represented by an activity-on-the-node network $G = (V,E)$ in which V denotes the set of vertices (nodes) representing the activities and E is the set of edges (arcs) representing the finish-start precedence relationships with zero time lag. The activities are numbered from *1* to n, where the dummy activities *1* and n mark the beginning and end of the project. The activities are to be performed without preemption. The fixed integer duration of an activity is denoted by d_i $(1 \leq i \leq n)$, its integer starting time by s_i $(1 \leq i \leq n)$ and its integer finishing time by f_i $(1 \leq i \leq n)$. There are K renewable resource types with r_{ik} $(1 \leq i \leq n,\ 1 \leq k \leq K)$ the constant resource requirement of activity i of resource type k and a_k the constant availability of resource type k. Conceptually, the RCPSP can be formulated as follows:

$$\min \quad f_n \qquad [1]$$

subject to

$$f_1 = 0 \qquad [2]$$

$$f_j - d_j \geq f_i \qquad (i,j) \in H \qquad [3]$$

$$\sum_{i \in S_t} r_{ik} \leq a_k \qquad t = 1,2,\ldots,f_n \ ; \quad k = 1,2,\ldots K \qquad [4]$$

where H denotes the set of pairs of activities indicating precedence constraints and S_t denotes the set of activities put in progress in time interval $]t\text{-}1,t]$: $S_t=\{i \mid f_i-d_i<t\leq f_i\}$. Eq. 2 assigns a completion time of 0 to the dummy start activity *1*. The precedence constraints given by Eq. 3 indicate that activity j can only be started if all predecessor activities i are completed. The resource constraints given in Eq. 4 indicate that for each time period $]t\text{-}1,t]$ and for each resource type k, the renewable resource amounts required by the activities in progress cannot exceed the resource availability. The objective function is given as Eq. 1. The project duration is minimized by minimizing the finishing time of the unique dummy ending activity n.

2.1 Optimal solution procedures and computational experience

The RCPSP, which as a generalization of the job-shop scheduling problem is NP-hard in the strong sense (Blazewicz et al. 1983), has been extensively studied in the literature. Previous research on optimal procedures basically involves the use of *mathematical programming* (Bowman 1959, Brand et al. 1964, Wiest 1964, Moodie and Mandeville 1966, Elmaghraby 1967, Pritsker et al. 1969, Patterson and Huber 1974, Patterson and Roth 1976, Deckro et al. 1991 and Icmeli and Rom 1996) and *implicit enumeration;* i.e. dynamic programming (Carruthers and Battersby 1966, Petroviç 1968) and branch-and-bound (Johnson 1967; Balas 1971; Schrage 1970; Davis and Heidorn 1971; Stinson et al. 1978; Talbot and Patterson 1978; Radermacher 1985; Christofides et al. 1987; Bartusch et al. 1988; Bell and Park 1990; Carlier and Latapie 1991; Demeulemeester and Herroelen 1992, 1996d; Mingozzi et al. 1995; Brucker et al. 1996; Carlier and Neron 1996). For comprehensive reviews we refer the reader to

Davis (1966, 1973), Herroelen (1972), Patterson (1984), Icmeli et al. (1993), Elmaghraby (1995), Herroelen and Demeulemeester (1995), and Özdamar and Ulusoy (1995). Over the past decade, considerable progress in the use of optimal procedures for the RCPSP has been reported on two de facto standard problem sets: the 110 test problems assembled by Patterson (1984) and the 480 test instances generated by Kolisch et al. (1995).

2.1.1 The Patterson problem set

Computational results obtained by Patterson (1984) on a problem set of 110 test problems (7-50 activities, 1-3 renewable resource types) using Fortran V codes and an Amdahl 470/V8, seemed to indicate that Talbot's solution procedure (Talbot and Patterson 1978) is effective whenever the average resource-constrainedness in a problem is low (the resource-constrainedness for resource k is defined as the average quantity of resource k when used by an activity divided by the availability of resource k, while the average resource-constrainedness is defined as the sum of the resource-constrainedness divided by the number of resources). Solving 97 problems in an average CPU time of 14.98 seconds, it would likely be the preferred solution approach where computer storage is a particularly limiting factor. The breadth-first branch-and-bound solution procedure of Stinson (Stinson et al. 1978) was found to be the fastest and only procedure capable of solving all the 110 test instances within the 5 minute CPU time limit (an average CPU time of 0.82 seconds). Stinson's code was claimed to be the best in those instances in which computer memory is not limiting. The Davis and Heidorn (1971) procedure solved 96 instances in an average time of 14.02 seconds and was the best in those instances for which the number of remaining feasible subsets was low.

Computational results obtained by Demeulemeester et al. (1994) cannot confirm the claim that the CAT algorithm (Christofides et al. 1987) is competitive with the Stinson et al. (1978) procedure. In addition, it has been shown (Demeulemeester et al. 1994) that the CAT algorithm may occasionally miss the optimum. Computational experience gained by Bell and Park (1990) and Carlier and Latapie (1991) indicate that their procedure does not perform well on the Patterson test problems in that it failed to generate the optimal solution for all the test problems. Carlier and Neron (1996) use bounds based on m-machine problems which are generated at the root of the search tree. Their branching scheme is based on so-called left-tight schedules in which activities are either scheduled at the beginning of a schedule or at its end. Results obtained on a subset of the Patterson problems, reveal the rather high computational cost of the procedure.

The depth-first *DH-procedure*, developed by Demeulemeester and Herroelen (1992), seems to be the fastest exact solution method for solving the RCPSP. Computational experience with the Patterson problem set, confirmed the DH-procedure to be, on the average, almost twelve times faster than the breadth-first procedure developed by Stinson et al. (1978), previously reported to be the most effective and efficient on this problem set. Their Turbo C code, running under DOS® on a 80486 processor with 25 MHz clock speed (IBM PS/2, Model 75), solved all the Patterson problems in an average CPU time of 0.073 seconds, with a maximum of 1.43 seconds and a standard deviation of 0.151 seconds. The

DOS®-version of the personal computer code allowed for an addressable memory of (less than) 640 Kb (kilobytes) in total, while, mainly for efficiency reasons, matrices used by the algorithm could not exceed a size of 64 Kb. Recent advances in 32 bit-compiler technology inspired the authors to revise and extend the procedure (subsequently referred to as the *DH1-procedure*) using a Microsoft Visual C++ 2.0® compiler under Windows NT 3.50® (Demeulemeester and Herroelen 1996d). This resulted in a speed boost by a factor of almost three on the 110 Patterson problems as compared to the code used for the 1992 paper. Using a 80486 processor running at 25 MHz, all 110 problems are solved in an average CPU time of 0.025 seconds, with a maximum of 0.23 seconds and a standard deviation of 0.026 seconds. Recent efforts to improve and repolish the code (subsequently referred to as the *DH2-procedure*) further reduced the computational effort to an average CPU time of 0.002 seconds to solve all 110 instances on a Dell personal computer, equipped with a Pentium Pro processor running at 200 MHz (maximum 0.02 seconds and a standard deviation of 0.004 seconds).

2.1.2 The 480 KSD test problems

Recent research by Kolisch, Sprecher and Drexl (1995) questioned the use of the 110 problem set and led to the development of *ProGen*, a network generator which allows for the generation of RCPSP problem instances which satisfy preset problem parameters. Computational experience on a total of 480 problem instances, generated on the basis of a full factorial design (32 activities including dummy start and end, 4 renewable resource types), revealed that the DH-procedure could optimally solve 428 instances in an average CPU time of 79.907 seconds, given a CPU time limit of one hour on an IBM PS/2 Model 55sx (80386sx processor, 15 MHz clockpulse). This finding inspired a number of authors (Kolisch et al. 1995, Mingozzi et al. 1995, Brucker et al. 1996) to claim that optimal solution procedures such as the DH-procedure cannot solve hard instances to optimality, even with a large amount of computing time.

Mingozzi et al. (1995) presented a new 0-1 linear programming formulation that requires an exponential number of variables, corresponding to all feasible subsets of activities that can be simultaneously executed without violating resource or precedence constraints. They present a tree search algorithm *BBLB3* based on this formulation which can solve the 52 hard KSD instances that could not be solved by the DH-procedure, while it is on the average 5 times slower on the Patterson test problems. They conclude that BBLB3 is competitive with the DH-procedure on hard instances, while it does not dominate DH on easier problems. Brucker et al. (1996) developed a branch-and-bound algorithm which performs a depth-first search on a binary search tree, the nodes of which correspond to so-called schedule schemes (sets of disjunctions, conjunctions, parallelity and flexibility relations). The authors develop various bounding and dominance rules and concepts of immediate selection. They report on computational experience with their algorithm on a subset of the Kolisch problems. The algorithm fails to terminate on 8 of the so-called hard instances, while it also fails to terminate on 20 of the so-called easy instances.

Demeulemeester and Herroelen (1996d) reported, however, that a close look at the 52 problems that could not be solved to optimality by the DH-procedure within the imposed time limit of one hour, indicated that the dominant factor which kept the procedure from finding the optimal solution, was not so much the computation time spent (as could be assumed from the results), but mainly the size of the computer memory that could be addressed. Exploiting the full potential of 32-bit programming, they developed the *DH1 procedure* which optimally solves the 480 Kolisch, Sprecher and Drexl (KSD) instances. Using an IBM PS/2 Model P75 with a 486 processor running at 25 MHz and a CPU time limit of 3600 seconds, 479 out of the 480 KSD instances were solved optimally in an average time of 12.331 seconds (maximum 2661.9 seconds with a standard deviation of 132.876 seconds). The remaining problem (KSD291) was solved within 3 hours of computation time. Moreover, a truncated version of the procedure yields excellent results. For many KSD instances the first solution found by the DH1-procedure is better than the one found by the popular MINSLK heuristic. Running the new DH-procedure for small amounts of time yields solutions which are very close to the optimum. Recent computational experience shows that the *DH2 procedure* solves all the 480 KSD instances in an average CPU time of 0.372 seconds on a Pentium Pro processor running at 200 MHz (maximum 50.97 seconds with a standard deviation of 2.744 seconds). These results constitute a new benchmark for the RCPSP. Moreover, the efficient and effective branching scheme, and many of the lower bound and dominance arguments have been extended to a wide and relevant variety of problem settings. They are elaborated on in the subsequent sections.

2.2 The DH- and the new DH-procedures

The *DH1-procedure* (Demeulemeester and Herroelen 1996d) is conceptually almost identical to the *DH-procedure* described in Demeulemeester and Herroelen (1992). It generates a search tree, the nodes of which correspond with partial schedules in which finish times temporarily have been assigned to a subset of the activities of the project. The partial schedules are feasible, satisfying both the precedence and resource constraints. *Partial schedules* PS_m are only considered at those time instants m which correspond to the completion time of one or more project activities. The partial schedules are constructed by semi-active timetabling. In other words, each activity is started as soon as it can within the precedence and resource constraints. A partial schedule PS_m at time m thus consists of the set of *temporarily* scheduled activities. Scheduling decisions are temporary in the sense that temporarily scheduled activities may be delayed as a result of decisions made at later stages in the search process. Partial schedules are built up starting at time 0 and proceed systematically throughout the search process by adding at each decision point subsets of activities, including the empty set, until a complete feasible schedule is obtained. In this sense, a complete schedule is a *continuation of a partial schedule.*

At every time instant m we define the *eligible set* E_m as the set of activities which are not in the partial schedule and whose predecessor activities have finished. These eligible activities can start at time m if the resource constraints are not violated. Demeulemeester and Herroelen (1992) have proven

two theorems which allow the procedure, at decision point m, to decide on which eligible activities must be scheduled by themselves, and which pair of eligible activities must be scheduled concurrently.

Theorem 1. *If at time m the partial schedule PS_m has no activity in progress and an eligible activity i cannot be scheduled together with any other unscheduled activity at any time instant $m' \geq m$ without violating the precedence and resource constraints, then there exists an optimal continuation of the partial schedule with the eligible activity i put in progress (started) at time m.*

Theorem 2. *If at time m the partial schedule PS_m has no activity in progress, and if there is an eligible activity i which can be scheduled concurrently with only one other unscheduled activity j at any time instant $m' \geq m$ without violating precedence or resource constraints, and if activity j is both eligible and not longer in duration than activity i, then there exists an optimal continuation of the partial schedule in which both activities i and j are put in progress at time m.*

If it is impossible to schedule all activities at time m, a *resource conflict* occurs which will produce a new branching in the branch-and-bound tree. The branches describe ways to resolve the resource conflict by deciding on which combinations of activities are to be delayed. A *delaying set D(p)* consists of all subsets of activities D_q , either in progress or eligible, the delay of which would resolve the current resource conflict at level p of the search tree. A *delaying alternative* D_q is minimal if it does not contain other delaying alternatives $D_v \in D(p)$ as a subset. Demeulemeester and Herroelen (1992) give the proof that in order to resolve a resource conflict, it is sufficient to consider only minimal delaying alternatives.

One of the minimal delaying alternatives (nodes in the search tree) is arbitrarily chosen for branching. The delay of a delaying alternative D_q is accomplished by adding a *temporal constraint* causing the corresponding activities to be delayed up to the *delaying point*, which is defined as the earliest completion of an activity in the set of activities in progress, that does not belong to the delaying alternative.

The branching scheme can best be illustrated on a small problem example. Assume that the set of activities {1,2,3,4} creates a resource conflict at decision point m and that the minimal delaying set is {{1},{2},{3,4}}. Assume that activity x is the earliest finishing activity among *2*, *3* and *4*, that activity y is the earliest finishing activity among the activities *1*, *3* and *4* and that activity z is the earliest finishing activity among the activities *1* and *2*. The resulting delaying alternatives are represented in Figure 1. The operator '<' denotes a *temporal* constraint, i.e. a delay up to the earliest finishing time of an activity in progress that does not belong to a delaying alternative.

The delayed activities are removed from the partial schedule and the set of activities in progress, and the algorithm continues by computing a new decision point. The search process continues until the dummy end activity has been scheduled. Every time such a complete schedule has been found, *backtracking* occurs: a new delaying alternative is arbitrarily chosen from the set of delaying alternatives *D(p)* at the highest level p of the search tree that still has some unexplored delaying

alternatives left, and branching continues from that node. When level zero is reached in the search tree, the search process is completed.

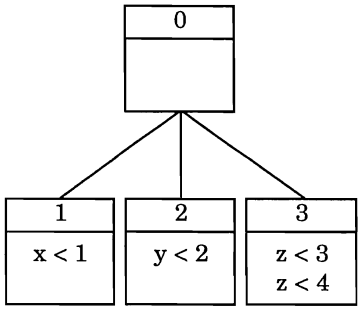

Figure 1. Minimal delaying alternatives

Two *dominance rules* are used to prune the search tree. The first one is a variation of the well-known left-shift dominance rule, and can be stated as follows:

Theorem 3. *If the delay of the delaying alternative at the previous level of the branch-and-bound tree forced an activity i to become eligible at time m, if the current decision is to start activity i at time m and if activity i can be left-shifted without violating the precedence or resource constraints (because activities in progress were delayed), then the corresponding partial schedule is dominated.*

The second dominance rule is based on the concept of a cutset. At every time instant m a *cutset* C_m is defined as the set of unscheduled activities for which all predecessor activities belong to the partial schedule Ps_m. The proof of the following theorem can be found in Demeulemeester (1992) and Demeulemeester and Herroelen (1992):

Theorem 4. *Consider a cutset C_m at time m which contains the same activities as a cutset C_k, which was previously saved during the search of another path in the search tree. If time k was not greater than time m and if all activities in progress at time k did not finish later than the maximum of m and the finish time of the corresponding activities in PS_m, then the current partial schedule PS_m is dominated.*

The original DH-procedure has been tested with three *lower bounding rules.* The well-known remaining critical path length bound *LB0* and critical sequence lower bound *LB1* (Stinson et al. 1978) are supplemented by an extended critical sequence lower bound *LB2* which is computed by repetitively looking at a path of unscheduled, non-critical activities in combination with a critical path. The *LB2* calculation starts by calculating the Stinson critical sequence lower bound. This allows to determine which activities cannot be scheduled within their slack time. Subsequently, all paths consisting of at least two unscheduled, non-critical activities, which start and finish with an activity that cannot be scheduled within its slack time, are constructed. A simple type of dynamic programming then allows for the calculation of the extended critical sequence bound for every non-critical path.

Subsequent research revealed that *LB0* outperformed the critical sequence lower bounds *LB1* and *LB2*, when used in combination with the cutset dominance pruning rule. As a result, both *LB1* and *LB2* have been removed from the procedure. Moreover, Mingozzi et al. (1995) have introduced a new lower bound *LB3*, based on a new mathematical formulation for the RCPSP and implemented by using a heuristic for solving a set packing problem. Demeulemeester and Herroelen (1996d) have incorporated their own version of *LB3* in the new DH-procedure based on the following heuristic. For each activity

$i \in A$ they determine its possible *companions,* i.e., the activities with which it can be scheduled in parallel, respecting both the precedence and resource constraints. All unscheduled activities i with a non-zero duration are then entered in a list L in non-decreasing order of the number of companions (non-increasing duration as a tie-breaker). The following procedure then yields a lower bound, *LB3*, for the partial schedule under consideration:

LB3:= the earliest completion time of the activities in progress;
while list L not empty **do**
    Take activity j on top of list L and determine its duration d_j;
    $LB3 := LB3 + d_j$;
    Remove activity j and its companions from list L;
enddo.

It is clear that other (more computationally intensive) heuristics can be used to calculate the lower bound *LB3*. The procedure described here is very fast and offers an excellent trade-off between tightness of the bound and the required computational effort. It generally improves the critical path lower bound, *LB0*, if there are pairs of activities that can be scheduled in parallel taking into consideration the precedence constraints only, but cannot be scheduled in this manner if resource constraints are taken into consideration.

In addition to the removal of *LB1* and *LB2* and the possibility to use both *LB0* and *LB3*, the DH1-procedure is the result of two additional changes, which have been made in order to gain on speed and to exploit the power of modern 32-bit compiler architecture. The major change has to do with a new coding scheme for the cutset dominance rule. Being limited to matrices of at most 64 Kb, the original DOS-version of the DH-procedure used four matrices for coding the dominance rule. Two matrices of 64 Kb were used to store cutsets with the necessary information to apply the dominance rule and two matrices of 16 Kb contained the pointers to the cutsets listed in the two 64 Kb matrices. The flat memory model of 32-bit programming, which allows for more efficient memory addressing and increased usable memory size, makes it possible to implement the cutset dominance rule using only two matrices: one very large cutset matrix contains the cutsets with the additional information, while a second matrix of 256 Kb was used to store the pointers to the cutsets in the cutset matrix. This implementation has two important advantages: more cutsets can be saved (increasing the impact of the cutset dominance rule) and the code becomes simpler (improving the speed of its application). A second change involved merging different resource types into one global resource type. This change became possible because integers automatically consist of 32 bits when 32-bit programming is used. Using, for instance, 8 bits for every resource type allows to combine four resource types into one 32-bit integer representing one global resource type. Combining several resource types into one (or a few) global resource type(s) leads to a definite speed-up of the code.

The logic of the *DH2-procedure* differs from the logic used by DH1 in the additional use of a new resource-based lower bound, and an improved immediate scheduling rule for putting eligible activities in progress, which replaces the rules described in Theorem 1 and Theorem 2. Other differences boil down to the use of 64 Mb of addressable memory, a more efficient coding of the cutset dominance rule

(involving a more effective way of storing efficient cutsets), some preprocessing and additional code polishing.

2.3 Problem complexity and the prediction of the computational requirements

As mentioned earlier, extensive computational experience with the optimal solution procedures for the RCPSP has been gained on different test sets of problem instances: the 110 Patterson problem set and the 480 KSD problem set. Ideally such a set should span the full range of complexity, from very easy to very hard problem instances. The generation of easy and hard problem instances, however, appears to be a very difficult task which heavily depends on the possibility to isolate the factors that precisely determine the computing effort required by the solution procedure used to solve a problem, and the calibration of the scale that characterizes such effort. The 110 test problems, assembled by Patterson (1984), are a collection from different sources and have not been generated by using a controlled design of specified problem parameters. The 480 KSD instances used by Kolisch et al. (1995) have been generated using the problem generator *ProGen* through the use of a controllable set of specified problem parameters. Recently, *ProGen* has been used to generate thousands of additional test instances, which have made it possible to gain additional important insight in the factors that seem to determine the complexity (in terms of the required computation time) of an RCPSP instance.

2.3.1 The relation between problem hardness and topological network structure

De Reyck and Herroelen (1996a) have generated five sets of 1000 RCPSP instances, each with 25 activities, a maximum number of predecessors, resp. successors set to 25, 3 resource types with a constant availability of 6 units, resource requirements drawn from the uniform distribution in the range [1,5], and activity durations drawn from the uniform distribution in the range [1,10]. In each of the five sets, the *coefficient of network complexity, CNC,* is set at a different value, varying from 1.5 in the first set to 2.5 in the fifth. Each RCPSP instance was then solved using the DH-procedure.

The CNC is undoubtedly one of the most popular 'measures of network complexity'. Introduced by Pascoe (1966) for activity-on-the-arc networks, and simply defined as the ratio of the number of arcs over the number of nodes, the measure has been adopted in a number of studies since then (Davis 1975, Talbot 1982, Patterson 1984, Kurtulus and Narula 1985, and Kolisch et al. (1995)). As observed by Kolisch et al. (1995), in the activity-on-the-node representation, 'complexity' has to be understood in the way that for a fixed number of activities (nodes), a higher complexity results in an increasing number of arcs and therefore in a greater connectedness of the network. A number of studies in the literature (Alvarez-Valdes and Tamarit 1989, Kolisch et al. 1995) seem to confirm that problems become easier with increasing values of the CNC, which makes the use of the CNC somewhat confounding (Elmaghraby and Herroelen (1980) already questioned the use of the CNC). Both Alvarez-Valdes and Tamarit (1989) and Kolisch et al. (1995) observe a negative correlation between the CNC and the required solution time for solving an RCPSP instance. De Reyck and Herroelen (1996a) reach the conclusion that it is very ambiguous to attach all explanatory power of problem complexity to the CNC. They observed a positive correlation between the CNC and the so-called *complexity index, CI.*

The complexity index, CI, is defined as the reduction complexity (Bein et al. 1992); i.e. the minimum number of node reductions sufficient (along with series and parallel reductions) to reduce a two-terminal acyclic network to a single edge. The CI-values for the instances used in the experiment range from 9 to 21. The authors found that the CI plays an important role in predicting the required computing effort for solving an RCPSP instance (the higher the CI, the easier the RCPSP instance) and that the CI outperforms the CNC as a measure of network complexity (the CNC explains nothing extra above what is already explained by the CNC). The reason for the strong explanatory power attributed to the CNC in previous experiments performed in the literature is probably due to the fact that when the CNC was varied, other parameters (such as the CI) were varied also, which led to problems with significant differences in 'complexity'.

In a subsequent experiment, De Reyck (1995b) again used *ProGen* to generate 4200 instances (25 activities, maximum number of start, resp. finish activities set to 5, maximum number of predecessors, resp. successors set to 25, 3 renewable resource types, activity durations and resource requirements drawn uniformly from the interval [1,10], CNC ranging from 1.2 to 2.5 and CI ranging from 1 to 17). Each instance was then solved using the DH1 procedure. Again the CI was found to have a strong impact on the required processing time whereas the CNC had no impact at all. In addition, Schwindt's conjecture (Schwindt 1995) could be confirmed that an estimator for the so-called *restrictiveness*, namely *RT*, is a good network complexity measure. De Reyck (1995b) has shown that RT is actually identical to the *order strength*, *OS*, one of the best complexity measures for generating and evaluating assembly line balancing problems (see De Reyck and Herroelen 1995). OS is defined as the number of precedence relations, including the transitive ones, divided by $n(n-1)/2$, where n denotes the number of activities (Mastor 1970). It is sometimes referred to as the *density* (Kao and Queranne 1992) and actually equal to 1 minus the *flexibility ratio*, defined by Dar-El (1973) as the number of zero entries in the precedence matrix divided by the total number of matrix entries. Using values of RT ranging from 0.15 to 0.70, De Reyck (1995b) reached the conclusion that RT absorbed the explanatory power of both the CNC and the CI, and that RT outperforms both measures.

2.3.2 The RCPSP and resource availability

De Reyck and Herroelen (1996a) have also tried to isolate the impact of the resource availability (or resource-constrainedness) on the required solution effort for solving the RCPSP. Elmaghraby and Herroelen (1980) conjectured that the relationship between the hardness of a problem (as measured by the CPU time required for its solution) and resource availability (scarcity) varies according to a bell-shaped curve. If resources are only available in extremely small amounts, there will be relatively little freedom in scheduling the activities. Hence, the corresponding RCPSP instance should be quite easy to solve. If, on the other hand, resources are amply available, the activities can be simply scheduled in parallel and the resulting project duration will be equal to the critical path length, leading again to a small computational effort.

Two of the best known parameters for describing resource availability (scarcity) that have been proposed in the literature are the resource factor and the resource strength. The *resource factor*, *RF*

(Pascoe 1966, Cooper 1976, Alvarez-Valdes and Tamarit 1989, Kolisch et al. 1995) reflects the average portion of resources requested per activity. If we have RF=1, then each activity requests all resources. RF=0 indicates that no activity requests any resource. The *resource strength, RS* (Cooper 1976) is redefined by Kolisch et al. (1995) as $\left(a_k - r_k^{\min}\right)/\left(r_k^{\max} - r_k^{\min}\right)$, where a_k is the toal availability of renewable resource type k, $r_k^{\min} = \max_{i=1,\ldots,n} r_{ik}$, and $r_k^{\max}$ is the peak demand of resource type k in the precedence-based earliest start schedule. Hence, with respect to one resource the smallest resource availability is obtained for RS=0. For RS=1, the problem is no longer resource-constrained. In their experiments, Kolisch et al. (1995) conclude (in contradiction with Alvarez-Valdes and Tamarit 1989) that RS has the strongest impact on solution times: the average solution time continuously increases with decreasing RS. De Reyck and Herroelen (1996a), however, could not confirm the continuous increase of the required solution time with decreasing RS but found a bell-shaped relationship, in accordance with the conjecture of Elmaghraby and Herroelen (1980).

Patterson (1976) defines the *resource-constrainedness, RC*, for each resource k as p_k/a_k, where a_k is the availability of resource type k and p_k is the average quantity of resource k demanded when required by an activity. The arguments for using RC and not RS as a measure of network complexity are that (a) RC is a 'pure' measure of resource availability in that it does not yet incorporate information about the precedence structure of a network, and (b) there are occasions where RS can no longer distinguish between easy and hard instances while RC continues to do so (for details, we refer to De Reyck and Herroelen 1996a). Again, De Reyck and Herroelen (1996a) are able to confirm a bell-shaped relationship between the CPU time and RC.

2.4 Branch-and-bound procedures for solving the RCPSP: conclusions

The fundamental conclusions which can be drawn from the reviewed research on branch-and-bound schemes for the RCPSP can be summarized as follows:

(i) a depth-first branch-and-bound search strategy based on resolving resource conflicts by delaying minimal subsets of activities is a clear favourite for optimally solving RCPSP instances;

(ii) the cutset dominance rule ranks amongst the most effective dominance pruning rules, especially if a sufficient amount of memory (e.g. 24 Mb) can be used for storing the cutsets;

(iii) the use of easy to compute and effective lower bounds (e.g. *LB3* and its possible variations; the new resource-based bound incorporated in DH2) has a strong impact on the computational cost;

(iv) it is extremely important to exploit the trade-off between the strength of the bounds or dominance rules used and the time required for their computation;

(v) truncated depth-first branch-and-bound procedures provide a suitable alternative to priority based heuristics such as MINSLK (the first solution obtained is often better than the one obtained by MINSLK; near-optimal solutions are obtained even if the truncated procedure is only allowed to run for a small amount of time, e.g. an average deviation on the 480 KSD problems of 0.575 % after 1 second);

(vi) sufficient attention should be devoted to efficient coding of the solution procedures used;

(vii) exploiting the full potential of 32-bit programming provided by recent compilers running on personal computer platforms such as Windows NT® and OS/2® may add considerably to the efficiency of the computer code used;

(viii) reproducable optimal benchmark results are available on the 110 Patterson problems and the 480 KSD problems. In order to avoid computational bias and to guarantee that procedures are validated on a relevant spectrum of problem complexity (the complexity of a problem instance is entwined to the procedure used to solve it), computational experience should be reported on the complete problem sets and should not be limited to selected problem subsets assumed to be "hard" or "easy".

3. The preemptive resource-constrained project scheduling problem (PRCPSP)

The *PRCPSP* allows activities to be preempted at integer points in time; i.e., the fixed integer duration d_i of an activity may be split in $j = 1,2,...,d_i$ duration units. Each duration unit j of activity i is then assigned an integer finish time $f_{i,j}$. The variable $f_{i,0}$ denotes the earliest time that an activity i can be started. As only finish-start relations with a time lag of zero are allowed, $f_{i,0}$ equals the latest finish time of all the predecessors of activity i. An activity i belongs to the set S_t of activities in progress in period $]t-1,t]$ if one of its duration units $j = 1,2,...,d_i$ finishes at time t (i.e., if $f_{i,j} = t$). The PRCPSP can now be conceptually formulated as follows (Demeulemeester 1992):

$$\min \quad f_{n,0} \qquad [5]$$

subject to

$$f_{i,d_i} \leq f_{j,0} \quad \text{for all } (i,j) \in H \qquad [6]$$

$$f_{i,j-1} + 1 \leq f_{i,j} \quad i = 1,\ldots,n; \quad j = 1,\ldots,d_i \qquad [7]$$

$$f_{1,0} = 0 \qquad [8]$$

$$\sum_{i \in S_t} r_{ik} \leq a_k \quad k = 1,\ldots,K; \; t = 1,\ldots,f_{n,0} \qquad [9]$$

The objective function (Eq. 5) minimizes the project length by minimizing the earliest start time of the dummy end activity which by assumption has a duration of 0. Eqs. 6 assure that all precedence relations are satisfied: the earliest start time of an activity j cannot be smaller than the finish time of the last unit of duration of its predecessor i. Eqs. 7 specify that the finish time for every unit of duration of an activity has to be at least one time unit larger than the finish time for the previous unit of duration. Activity *1* is assigned an earliest start time of 0 through Eq. 8, while Eqs. 9 stipulate the resource constraints.

Slowinski (1980) and Weglarz (1981) have presented optimal solution procedures for the case of continuous processing times for the different activities. Davis and Heidorn (1971) developed an implicit

enumeration scheme based on the splitting of activities in unit duration tasks. Kaplan (1988, 1991) presents a dynamic programming formulation and suggests a solution by a reaching procedure.

The DH-procedure has been extended to the PRCPSP (Demeulemeester and Herroelen 1996b). In order to do so it is assumed that only two dummy activities exist in the project: the dummy start and the dummy end. This is caused by the time incrementing scheme used, which augments the decision points by one time unit at a time. In addition a distinction is made between *activities* and *subactivities*. At the start of the procedure we create a new project network in which all activities are replaced by one or more subactivities. The dummy start and end activities are replaced by dummy start and end subactivities with a duration of 0. All other activities are split into subactivities, their number being equal to the duration of the original activity, each having a duration of 1 and resource requirements that are equal to those of the original activity. Demeulemeester and Herroelen (1996b) prove that in order to solve the PRCPSP, it is sufficient to construct partial schedules by semi-active timetabling at the level of the subactivities.

An *eligible activity* is defined as an activity for which one of the subactivities is eligible. An *unfinished activity* is an activity for which not all subactivities have been scheduled. Denote the z unfinished subactivities of unfinished activity i at time t as $i_1, i_2, ..., i_z$. We say that activity i is scheduled immediately at time t if all its remaining subactivities i_x $(x = 1,...,z)$ are scheduled such that $f_{ix} = t + x$.

Theorems 1 and 2 stated above for the RCPSP can now be extended in the following way.

Theorem 5. *If for a partial schedule PS_m at time instant m there exists an eligible activity i that cannot be scheduled together with any other unfinished activity j at any time instant $m' \geq m$ without violating the precedence or resource constraints, then an optimal continuation of PS_m exists with all remaining subactivities $i_1, i_2, ..., i_z$ of activity i scheduled immediately at time m.*

The reader should notice that it is not necessary to check whether an activity is in progress or not at time m. The scheduling of an activity at the previous decision point does not imply that if that activity was not completed, the same activity should also be scheduled at the current decision point. The preemption condition allows to forget the scheduling decisions in previous periods and to consider only those possibilities implied by the set of eligible subactivities.

Theorem 6. *If for a partial schedule PS_m at time instant m there exists an eligible activity i that can be scheduled together with only one other unfinished activity j at any time instant $m' \geq m$ without violating the precedence or resource constraints and if activity j is eligible, then an optimal continuation of PS_m exists with all remaining subactivities of activity i scheduled immediately and with as many subactivities of activity j as possible scheduled concurrently with the subactivities of activity i.*

The reader will have noticed that no test needs to be performed to check whether the remaining duration of activity j is larger than that of activity i. Indeed, if the remaining duration of activity j is larger, as many subactivities of activity j will be scheduled as there are unscheduled subactivities in

activity *i*. If, however, the remaining duration of activity *j* is smaller or equal, all remaining subactivities of activity *j* will be scheduled concurrently with those of activity *i*.

Demeulemeester and Herroelen (1996b), are able to prove the following dominance rule which very much resembles the cutset dominance rule stated earlier for the RCPSP.

Theorem 7. *Consider a partial schedule PS_m at time m. If there exists a partial schedule PS_k that was previously saved at a similar time m and if PS_m is a subset of PS_k, then the current partial schedule PS_m is dominated.*

Demeulemeester and Herroelen (1996b) also show that it is sufficient to consider only minimal delaying alternatives in order to resolve resource conflicts. In addition, they have shown that all three lower bounds discussed earlier (*LB0*, *LB1* and *LB2*) remain applicable, at the trade-off of increased computational requirements. Therefore, they only included *LB0* in the code. *LB3*, which is extendable to the PRCPSP but was only developed very recently, could not be included at the time the code was written.

As was already mentioned before, the literature on the PRCPSP is almost void and very little computational experience is available. Demeulemeester and Herroelen (1996b) have programmed their procedure in Turbo C Version 2.0 for a personal computer IBM PS/2 Model 70. On the same 41 Patterson test problems used by Kaplan (1988, 1991) and using a similar PC running at 16 MHz, it finds the optimal solution in an average CPU time of 4.9863 seconds with a standard deviation of 9.2932 seconds, while the Kaplan code requires an average of 425 seconds and a standard deviation of 713 seconds, respectively. Using a personal computer IBM PS/2 running at 25 MHz, they have tested their algorithm on all 110 Patterson test problems. All problems could be solved within 5 minutes of CPU time, requiring an average of 6.8985 seconds and a standard deviation of 25.8149 seconds.

Demeulemeester (1992) has extended the code for the PRCPSP with variable resource availabilities. In that case, Theorems 5 and 6 no longer apply. A total of 107 out of the 110 Patterson test problems, modified by Simpson and Patterson (1996) to incorporate variable resource availabilities, could be solved on an average computation time of 12.6321 seconds and a standard deviation of 36.9071 seconds.

4. Project scheduling under generalized precedence relations

A lot of research efforts have been directed towards relaxing the strict precedence assumption of CPM/PERT. The resulting types of precedence relations are often referred to as MPM (Metra Potential Method) precedence constraints (Kerbosch and Schell 1975, Zhan 1994), precedence diagramming (Moder et al. 1983), time windows (Bartusch et al. 1988), minimal and maximal time lags (Brinkmann and Neumann 1994, Neumann and Schwindt 1995, Schwindt 1995, Franck and Neumann 1996, Neumann and Zahn 1996, Schwindt and Neumann 1996), and generalized precedence constraints (Wikum et al. 1994). In accordance with Elmaghraby and Kamburowski (1992), we denote them as generalized precedence relations (GPRs) and distinguish between start-start (*SS*), start-finish (*SF*), finish-start (*FS*) and finish-finish (*FF*).

GPRs can specify a minimal or maximal time lag between any pair of activities. A *minimal* time lag specifies that an activity can only start (finish) when the predecessor activity has already started (finished) for a certain time period. A *maximal* time lag specifies that an activity should be started (finished) at the latest a certain number of time periods beyond the start (finish) of another activity.

4.1. The generalized resource-constrained project scheduling problem (GRCPSP)

Demeulemeester and Herroelen (1996a) have extended the DH-procedure to the case of *minimal* time lags, activity release dates and deadlines and variable resource availabilities. The resulting problem, which they denote as the GRCPSP, can be conceptually formulated as follows:

$$\min \quad f_n \qquad [10]$$

subject to

$$f_i - d_i + SS_{ij} \leq f_j - d_j \quad \text{for all } (i,j) \in H_1 \qquad [11]$$

$$f_i - d_i + SF_{ij} \leq f_j \quad \text{for all } (i,j) \in H_2 \qquad [12]$$

$$f_i + FS_{ij} \leq f_j - d_j \quad \text{for all } (i,j) \in H_3 \qquad [13]$$

$$f_i + FF_{ij} \leq f_j \quad \text{for all } (i,j) \in H_4 \qquad [14]$$

$$f_1 = 0 \qquad [15]$$

$$f_i - d_i \geq g_i \quad i = 1,2,\ldots,n \qquad [16]$$

$$f_i \leq h_i \quad i = 1,2,\ldots,n \qquad [17]$$

$$\sum_{i \in S_t} r_{kt} \leq a_{kt} \quad k = 1,2,\ldots,K;\ t = 1,2,\ldots,f_n \qquad [18]$$

where

H_1 = set of pairs of activities indicating start-start relations with a time lag of SS_{ij}

H_2 = set of pairs of activities indicating start-finish relations with a time lag of SF_{ij}

H_3 = set of pairs of activities indicating finish-start relations with a lag of FS_{ij}

H_4 = set of pairs of activities indicating finish-finish relations with a lag of FF_{ij}

g_i = ready time of activity i

h_i = due-date of activity i

a_{kt} = availability of resource type k during period $]t\text{-}1,t]$

The objective function (Eq. 10) is to minimize the project duration by minimizing the finish time of the unique dummy end activity n. Eqs. 11-14 ensure that the various types of precedence constraints are satisfied. Eq. 15 assigns the dummy start activity *1* a completion time of 0. Eqs. 16 guarantee that the ready times are respected, while Eqs. 17 guarantee that no due-dates are violated. Eqs. 18 specify that the resource utilization during any time interval $]t\text{-}1,t]$ does not exceed the resource availability levels during that time interval for any of the resource types.

In order to extend the DH-procedure to the GRCPSP, all precedence constraints are converted to finish-start precedence relations using the following conversion formula:

$$FS'_{ij} = \max\left\{SS_{ij} - d_i, SF_{ij} - d_i - d_j, FS_{ij}, FF_{ij} - d_j\right\} \quad [19]$$

The *ready time* g_i of an activity i can easily be transformed into a finish-start relation between the dummy start activity *1*, which starts and finishes at time 0, and activity i itself:

$$FS''_{1i} = \max\left\{g_i, FS'_{1i}\right\} \quad [20]$$

Coping with *deadlines* h_i is somewhat more involved. For every activity j a latest allowable start time ls_j has to be computed such that whenever this activity j is delayed to start later than ls_j, the deadline of this activity or of one of its direct or indirect successors is exceeded even if all subsequent activities were scheduled as soon as possible without considering the resource constraints. Consequently, if during the branch-and-bound procedure an activity j is assigned an early start time s_j that exceeds its latest allowable start time ls_j, backtracking can occur as no feasible solution can be found by continuing the search from this partial schedule.

As before, S_t is defined as the *set of activities in progress* during the time interval]*t-1,t*], PS_t as the *partial schedule* which contains the set of activities that have been assigned a finish time at time *t*, and the *cutset* C_t as the set of all unscheduled activities whose predecessors all belong to the partial schedule PS_t. The *eligible set* E_t then denotes the set of all activities that belong to the cutset C_t and that can start at time *t*. The precise time instant at which these sets are defined will be clear from the context, hence, the subscripts will be omitted for simplicity of notation.

The search process starts by adding the dummy start activity *1* to *S* and *PS* with a finish time f_1=0. All activities i that have activity *1* as a single predecessor are added to the cutset and are assigned an early start time, based on the precedence relations FS''_{1i} (which include the ready times). The next decision point *m* is then computed as the smallest early start time of any activity in the cutset. The activities in the cutset that can start at time *m* are added to the eligible set *E*. All activities in *S* that complete before time *m* are deleted from *S* and all activities in *E* are scheduled: they are added to *S* and *PS* and are assigned a finish time that equals the sum of the decision point *m* and the duration of the activity involved. The cutset is updated. If due to resource constraints it is impossible to schedule all activities in *E* concurrently, a resource conflict occurs. Such a conflict will produce a new branching in the branch-and-bound tree at level *p*: the branches describe ways to resolve the resource conflict; i.e., decisions about which combinations of activities are to be delayed.

A *delaying alternative* D_q is defined as the set of activities that belong to *S*, the delay of which would resolve the resource conflict that occurred at level *p* of the solutions tree and for which it holds that if an activity belongs to D_q all its direct and indirect successors that belong to *S* are also included in D_q. In order to simplify the construction process of the delaying alternatives, a precedence relation is

added for every activity that can be partially overlapped with one of its indirect successors. As such, only the direct successors need to be examined in order to satisfy the second condition. The *delaying set D(p)* then consists of all possible delaying alternatives D_q that resolve the resource conflict at level p of the branching tree. For each delaying alternative D_q the *delaying point* w_q is computed as the earliest time at which either the resource availability changes, or an activity that belongs to (S-D_q) finishes, or one of the unscheduled activities that has no predecessor in D_q could finish if all unscheduled activities were scheduled as soon as possible. A *precedence based lower bound* L_q is then calculated by adding the maximal remaining critical path length of any of the activities that belong to D_q to the delaying point w_q. The delaying alternative with the smallest lower bound is chosen (ties are broken arbitrarily) and these activities are removed from S and PS (as well as all completed successors of one of these activities). All other delaying alternatives are stored for backtracking purposes. The cutset is updated and the process of constructing the eligible set, adding it to S and PS and branching whenever resource conflicts occur is repeated until a solution to the problem is found or until it can be shown that by branching from this node only infeasible solutions or dominated solutions could be generated. When this happens the procedure backtracks.

Demeulemeester and Herroelen (1996a) prove that the partial schedules may be constructed by semi-active timetabling. In addition they show that it is sufficient to consider only minimal delaying alternatives in order to resolve a resource conflict. Last but not least, they extend the left-shift and cutset dominance rules. They also show that the critical sequence bound *LB1* and the extended critical sequence bound *LB2* cannot be extended, leaving the remaining critical path length *LB0* as a possible lower bound (again *LB3* can be extended to the GRCPSP but was not yet known when writing the code).

The literature on the GRCPSP is very limited and a standard set of test problems has not yet been established. Demeulemeester and Herroelen (1996a) have coded the *GDH-procedure* in Turbo C Version 2.0 for IBM PS/2 computers with 80486 processor operating at 25 MHz (or compatibles). The procedure was then tested on the 110 Patterson test problems as modified by Simpson and Patterson (1996) to incorporate variable resource availabilities. The GDH-procedure could find the optimal solution for all 110 problems with constant resource availabilities in an average of 0.1446 seconds and a standard deviation of 0.3027 seconds. For the problems with variable resource availabilities,the GDH-procedure, when given a time limit of 10 minutes, could optimally solve 109 out of the 110 problems to optimality in an average CPU time of 8.1065 seconds (vis-à-vis 100.85 seconds required on average by Simpson's serial procedure to solve 97 problems and 96.63 seconds required on average by Simpson's parallel procedure to solve 98 problems) and a standard deviation of 57.775 seconds (vis-à-vis 199.62 seconds for Simpson's serial procedure and 195.90 seconds for Simpson's parallel procedure). When the code is allowed to run to completion, all problems are solved in an average of 60.1561 seconds and a standard deviation of 548.9009. As such, the GDH-procedure seems to be a very efficient and

effective exact solution procedure for the GRCPSP. In addition, the computational experience obtained indicates that moderate changes in the ready times or in the resource availabilities do not have a significant impact on the computation times. The introduction of due-dates significantly reduces the solution time required. Allowing activity overlaps (negative FS_{ij} values) causes a strong increase in the required computation time.

In the next section it is shown that a modification of the delaying scheme allows the DH-procedure to be extended to the case of resource-constrained project scheduling with minimal *and* maximal time lags.

4.2 The resource-constrained project scheduling problem with generalized precedence relations (RCPSP-GPR)

The resource-constrained project scheduling problem with generalized precedence relations (RCPSP-GPR) allows for start-start, finish-start, start-finish and finish-finish constraints with minimal *and* maximal time lags. The minimal and maximal time lags between two activities i and j have the form:

$$s_i + SS_{ij}^{\min} \le s_j \le s_i + SS_{ij}^{\max}$$
$$s_i + SF_{ij}^{\min} \le f_j \le s_i + SF_{ij}^{\max}$$
$$f_i + FS_{ij}^{\min} \le s_j \le f_i + FS_{ij}^{\max}$$
$$f_i + FF_{ij}^{\min} \le f_j \le f_i + FF_{ij}^{\max}$$

The different types of GPRs can be represented in a *standardized form* by reducing them to just one type, e.g. the minimal start-start precedence relations, using the following transformation rules (Bartusch et al. 1988):

$$s_i + SS_{ij}^{\min} \le s_j \quad \Rightarrow \quad s_i + l_{ij} \le s_j \quad (l_{ij} = SS_{ij}^{\min})$$
$$s_i + SS_{ij}^{\max} \ge s_j \quad \Rightarrow \quad s_j + l_{ji} \le s_i \quad (l_{ji} = -SS_{ij}^{\max})$$
$$s_i + SF_{ij}^{\min} \le f_j \quad \Rightarrow \quad s_i + l_{ij} \le s_j \quad (l_{ij} = SF_{ij}^{\min} - d_j)$$
$$s_i + SF_{ij}^{\max} \ge f_j \quad \Rightarrow \quad s_j + l_{ji} \le s_i \quad (l_{ji} = d_j - SF_{ij}^{\max})$$
$$f_i + FS_{ij}^{\min} \le s_j \quad \Rightarrow \quad s_i + l_{ij} \le s_j \quad (l_{ij} = d_i + FS_{ij}^{\min})$$
$$f_i + FS_{ij}^{\max} \ge s_j \quad \Rightarrow \quad s_j + l_{ji} \le s_i \quad (l_{ji} = -d_i - FS_{ij}^{\max})$$
$$f_i + FF_{ij}^{\min} \le f_j \quad \Rightarrow \quad s_i + l_{ij} \le s_j \quad (l_{ij} = d_i - d_j + FF_{ij}^{\min})$$
$$f_i + FF_{ij}^{\max} \ge f_j \quad \Rightarrow \quad s_j + l_{ji} \le s_i \quad (l_{ji} = d_j - d_i - FF_{ij}^{\max})$$

Conceptually, the RCPSP-GPR can then be formulated as follows:

$$\min \quad s_n \tag{21}$$

subject to

$$s_i + l_{ij} \leq s_j \qquad \forall (i,j) \in E \tag{22}$$

$$\sum_{i \in S_t} r_{ik} \leq a_{kt} \qquad k = 1,2,\ldots,K; \quad t = 1,2,\ldots,T \tag{23}$$

$$s_1 = 0 \tag{24}$$

$$s_i \in N \qquad i = 1,2,\ldots,n \tag{25}$$

where S_t is the set of activities in progress in time period]t-1,t]. The objective function given in Eq. 21 minimizes the project duration, given by the starting time (or finishing time: $d_n = 0$) of the dummy activity n. The precedence constraints are denoted in standardized form by Eqs. 22. Eqs. 23 represent the resource constraints. The resource requirements are assumed to be constant over time, although this assumption can be relaxed using GPRs without having to change the solution procedures (Bartusch et al. 1988). Eq. 24 forces the dummy start activity to begin at time zero and Eqs. 5 ensure that the activity starting times assume nonnegative integer values. Once started, activities run to completion.

The RCPSP-GPR is known to be strongly NP-hard, and even the feasibility problem, i.e. the problem of testing whether a RCPSP-GPR instance has a feasible solution, is NP-complete (Bartusch et al. 1988). To the best of our knowledge, the only optimal solution procedure in the open literature for the RCPSP-GPR is the branch-and-bound algorithm of Bartusch et al. (1988). De Reyck and Herroelen (1996b) present a new branch-and-bound procedure for the RCPSP-GPR based on the concepts of minimal delaying alternatives as developed by Demeulemeester and Herroelen (1992) for the RCPSP and adapted by Icmeli and Erengüç (1996) for the RCPSP with discounted cash flows.

The nodes in the search tree represent the initial project network, described by a distance matrix $D = [d_{ij}]$, extended with extra (zero-lag finish-start) precedence relations to resolve a resource conflict present in the parent node, which results in an *extended* distance matrix. Nodes which represent time-feasible (no violated maximal time lags) but resource-infeasible project networks and which are not fathomed by any of the node fathoming rules described below lead to a new branching. Therefore each (undominated) node represents a time-feasible, but not necessarily resource-feasible project network.

Resource conflicts are resolved using the concept of *minimal delaying alternatives.* Each of these minimal delaying alternatives is then delayed (enforced by extra precedence relations $i < j$) by each of the activities also belonging to the *conflict set* S_{t^*}, the set of activities in progress in period]t^*-1,t^*] (the period of the *first* resource conflict), but not belonging to the delaying alternative. This strategy is similar to the delaying strategy used by Demeulemeester and Herroelen (1992) for the RCPSP. As the RCPSP can be solved using semi-active timetabling to construct the partial schedules, activities belonging to the minimal delaying alternative can be delayed by the activity in S_{t^*} which terminates at the earliest time instant after the current decision point. In the RCPSP-GPR, however, this delaying strategy cannot be used because of the presence of maximal time lags, which make semi-active timetabling inappropriate. The same problem occurs in the RCPSP with discounted cash flows (RCPSP-

DC) discussed later in this text. Icmeli and Erengüç (1996) have modified the delaying scheme of Demeulemeester and Herroelen (1992) for the RCPSP-DC. A similar scheme can be used for the RCPSP-GPR.

There are several *delaying modes* for delaying a delaying alternative. Each delaying alternative can give rise to several delaying modes, possibly one for each activity in S_{t*} which is not an element of the delaying alternative. In general, the *delaying set D*, the set of all minimal delaying alternatives, is equal to: $D = \left\{ D_d \middle| D_d \subset S_t*, \forall \text{ resource } k : \sum_{i \in S_t*} r_{ik} - \sum_{i \in D_d} r_{ik} \leq a_k \text{ and } \forall D_{d'} \in \mathrm{D} : D_{d'} \not\subset D_d \right\}$.

The set of minimal delaying modes equals: $M = \left\{ M_m \middle| M_m = \left\{ k < D_d \right\}, k \in S_t* \setminus D_d, D_d \in D \right\}$.

Activity k is called the *delaying activity*: $k < D_d$ implies that $k < l$ for all $l \in D_d$.

For the example described in Section 2.2 (see Figure 1), the 8 resulting delaying modes are depicted in Figure 2.

Each minimal delaying mode is examined for time-feasibility and, if feasible, evaluated by computing the critical path based lower bound *LB0*. Each time-feasible minimal delaying mode with a lower bound *LB0* ≤ *T* is then considered for further branching, which occurs from the node with smallest *LB0*. If the node represents a project network in which a resource conflict occurs, a new branching occurs. The procedure is of the depth-first type, i.e. branching occurs until at a certain level in the tree, there are no delaying modes left to branch from. Then, the procedure backtracks to the previous level in the search tree and reconsiders the other delaying modes (not yet branched from) at that level. The procedure stops when it backtracks to level 0.

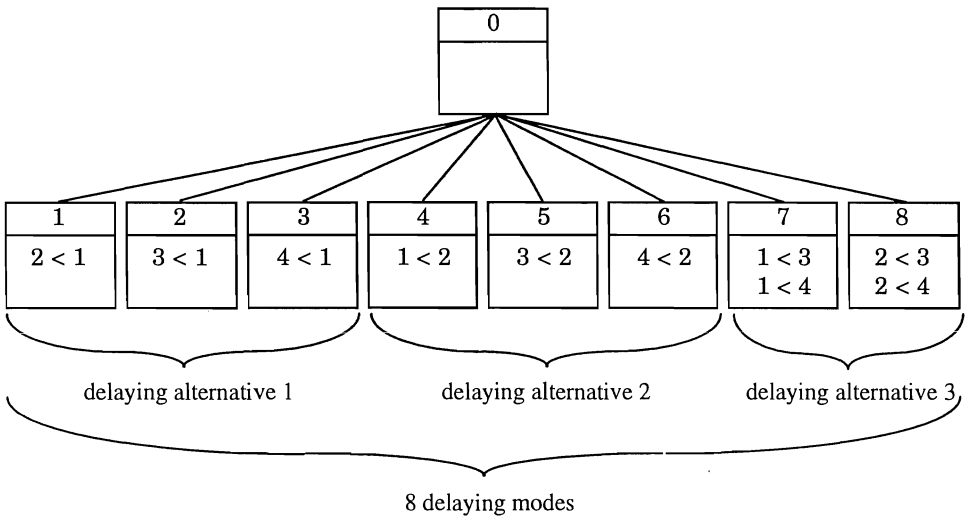

Figure 2. The concept of delaying modes

Nodes are fathomed when they represent a time-infeasible network or when *LB0* exceeds *T*. The following two dominance rules rely on the identification of redundant delaying alternatives and redundant delaying modes:

Theorem 8. *If there exists a minimal delaying alternative D_d with activity $i \in D_d$ but its real successor $j \notin D_d$ ($d_{ij} \geq 0$), D_d can be extended with activity j. If the resulting delaying alternative becomes non-minimal as a result of this operation it may be eliminated from further consideration.*

Theorem 9. *When a minimal delaying alternative D_d gives rise to two delaying modes with delaying activities i and j, the latter mode is dominated by the former iff $d_{ij} + d_j \geq d_i$.*

In addition, the following precedence subset dominance rule is incorporated in the procedure:

Theorem 10. *If the set of added precedence constraints which leads to the project network (in the form of an extended distance matrix) in node x contains as a subset another set of precedence constraints leading to the project network (extended distance matrix) in a previously examined node y in another branch of the search tree, node x can be fathomed.*

Before initiating the branch-and-bound procedure, the solution space can be reduced by the following preprocessing rule:

Theorem 11. *If $\exists\, i, j \in V$ and resource type k for which $r_{ik} + r_{jk} > a_k$ and $-d_j < d_{ij} < d_i$, we can set $l_{ij} = d_i$ without changing the optimal solution of the RCPSP-GPR.*

The procedure of Demeulemeester and Herroelen (1996d) for computing *LB3* can be extended to the RCPSP-GPR by changing the calculation of the companions of the activities. In the RCPSP-GPR, two activities i and j between which a precedence relation exists, are companions if the resource requirements of both activities do not exceed the resource availability for any resource type, and if both $d_{ij} < d_i$ and $d_{ji} < d_j$. Instead of removing an activity j from the list L when a companion i is taken from the list, only part of activity j is removed from the list. The logic behind this reasoning relies on both a duration and time lag argument. The duration argument goes as follows: When an activity i is scheduled, a companion j can be scheduled in parallel with i. However, if $d_i < d_j$, only a part of activity j can be scheduled in parallel with i. Therefore, a part of activity j (with remaining duration $d'_j = d_j - d_i$) can be left in L. Initially, all d'_j are equal to d_j. The time lag argument involves adjusting the part of activity j which has to be removed when a companion i is taken from the list, by incorporating precedence relations between i and j. When deciding on how much to remove from activity j, we have to look at each combination of minimal and maximal time lags between two activities i and j.The calculation of *LB3'* can now be summarized as follows:

$LB3' = 0$;
while L not empty **do**
  take the first activity i in L and remove it;
  $LB3' = LB3' + d_i^r$;
  for every companion j of i **do**
    if $d_{ij} > 0$ **then** $d_j^r = d_j^r - (d_i - d_{ij})$;
    elseif $d_{ji} > 0$ **then** $d_j^r = d_j^r - \min\{d_j - d_{ji}, d_i\}$;
    else $d_j^r = d_j^r - d_i$;
    endif

if $d_j^r \leq 0$, remove activity j from L;
enddo
enddo

The procedure has been programmed in Microsoft® Visual C++ 2.0 under Windows NT for use on a Pentium-60 personal computer with 16 Mb of internal memory. In order to validate the branch-and-bound procedure, 550 RCPSP-GPR instances were generated based on the problem set for the RCPSP assembled by Patterson (1984). When all fathoming rules are included, the average computation time and the number of nodes in the search tree are minimal, and so are their variances. A detailed analysis reveals that the percentage of maximal precedence relations, their tightness and the percentage of precedence relations that allow for activity overlaps have a significant impact on the computational effort. The higher the number of maximal time lags and the higher the number of minimal time lags that allow for activity overlaps, the more efficient the procedure. In order to test the performance of a truncated version of the procedure, an experiment was performed in which the procedure was run until (a) the first feasible solution was found, (b) for 1 second and (c) for 10 seconds. The average deviation from the optimum for these three cases was 4.6%, 0.8% and 0.1% respectively. When many maximal time lags are present and the minimal time lags allow for activity overlaps, the average deviation decreases to 3.75%, 0% and 0% respectively.

Recently, De Reyck and Herroelen (1996c) report on new computational experience on three different problem sets generated using the problem generator *ProGen/max* developed by Schwindt (1995). The first set (Schwindt 1996) consists of 1080 instances involving 100 activities and 5 resource types, satisfying a variety of preset parameters. The second set of 100 activity problems consists of 1440 instances generated by Franck and Neumann (1996). De Reyck and Herroelen (1996c) use a third set of 7200 instances ranging in problem size from 10 up to 100 activities with a requirement for 5 resource types. They show that a truncated version of the procedure outperforms a combination of the best heuristic procedures available (Neumann and Zahn 1996, Brinkmann and Neumann 1994, Franck and Neumann 1996, Schwindt and Neumann 1996) in less than 2 seconds of computation time on average, whereas the heuristics themselves consume much more CPU time.

5. Maximizing the net present value in project networks

In recent years, a number of publications have dealt with the project scheduling problem under the *irregular* objective of maximizing the net present value *(npv)* of the project. The majority of the contributions assume a completely *deterministic* project setting, in which all relevant problem data, including the various cash flows, are assumed known from the outset. Research efforts have led to optimal procedures for the *unconstrained* project scheduling problem, where activities are only subject to precedence constraints. In addition, numerous efforts aim at providing optimal or suboptimal solutions to the project scheduling problem under various types of resource *constraints*, using a rich variety of often confusing assumptions with respect to network representation (activity-on-the-node versus activity-on-the-arc), cash flow patterns (positive and/or negative, event-oriented or activity-based), and resource constraints (capital constrained, different resource types, materials considerations,

time/cost trade-offs). A number of efforts focus on the simultaneous determination of both the amount and timing of payments. Last, a modest start has been taken in tackling the *stochastic* aspects of the scheduling problem involved. For a recent review of the vast literature and a categorization of the solution procedures, we refer the reader to Herroelen et al. (1996).

5.1 The max-npv problem

Demeulemeester et al. (1996b) have recently developed a very efficient procedure for solving the deterministic unconstrained max-npv problem. The project is represented by an AoN network $G=(V,E)$ where the set of nodes, V, represents activities and the set of arcs, E, represents finish-start precedence constraints with a time lag of zero. We assume, without loss of generality, that there is a single dummy start node *1* and a single dummy end node $n = |V|$. The problem is unconstrained in that no constraints are imposed on the use of resources. The activities have a fixed duration, d_i, $i=1,2,...,n$, and the performance of each activity involves a series of cash flow payments and receipts throughout the activity duration. A terminal value of each activity upon completion can be calculated by compounding the associated cash flows to the end of the activity as follows:

$$C_i = \sum_{t=1}^{d_i} F_{it}\ e^{\alpha(d_i - t)} \qquad [26]$$

where

d_i = duration of activity i (a fixed integer number of periods)

C_i = terminal value of cash flows in activity i at its completion

F_{it} = cash flows for activity i in period t, $t = 1,2,...,d_i$

α = discount rate

A conceptual formulation of the deterministic *unconstrained max-npv problem* can now be formulated as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} q_{f_i}\ C_i \qquad [27]$$

subject to

$$f_i \leq f_j - d_i \qquad \text{for all } (i,j) \in H \qquad [28]$$

$$f_n \leq T \qquad [29]$$

with C_i and d_i as defined earlier and where

f_i = completion time of activity i (integer variable)

q_t = factor for discounting over t periods to period zero; i.e., $q_t = exp(-\alpha t)$

T = project deadline

The objective (Eq. 27) is to maximize the net present value of the project. The constraint set given in Eqs. 28 maintains the finish-start precedence relations among the activities. The final constraint (Eq. 29) limits the project duration to a negotiated project deadline.

The algorithm starts by computing the earliest completion time for the activities based on traditional forward pass critical path calculations and determines the corresponding *early tree* in the network which spans all activities (nodes) scheduled at their earliest completion times and which corresponds to a feasible solution with a project duration equal to the critical path length (the arcs of the early tree denote the binding precedence relations). The algorithm then builds the *current tree* by delaying, in reverse order, all nodes with a negative cash flow as much as possible within the early tree; i.e., by linking them to their earliest starting successor. Using the dummy node *1* as the search base, the algorithm will enter a *recursive search* (***recursion(1)*** in the write-up given below) of the current tree to identify partial trees that might be shifted forwards (away from time zero) in order to increase the *npv* of the project. Due to the structure of the recursive search it can never happen that a backward shift (towards time zero) of a partial tree can lead to an increase in the *npv* of the project: any partial tree that is not scheduled at its earliest starting point has a negative *npv* and should be scheduled as late as possible. When a partial tree is the subject of a forward shift, the algorithm computes its minimal displacement interval and updates the current tree. Upon a shift, the algorithm repeats the recursive search on the current tree associated with the new feasible solution. During the search, it is possible that the current tree disconnects into two parts, one part being shifted forward till it hits the deadline. If this happens, repetitively performing recursion(1) will only optimize the tree connected to node *1*. In order to optimize the second tree we have to perform a similar recursion starting in node *n* (***recursion(n)*** in the write-up). The algorithm stops when no partial trees can be shifted that increase the *npv* of the project.

Using *PT* to denote a partial tree, *DC* to denote the corresponding discounted net cash flow and *A* to denote the set of already examined nodes, the recursive algorithm can be written as follows:

Step 1:

- Compute the *early tree* using standard critical path calculations.
- In reverse order, delay all nodes with a negative cash flow as much as possible within the early tree; i.e., link them to their earliest starting successor.
- Make the resulting tree the *current tree* with activity completion times $(f_1, f_2, \ldots, f_n)$.

Step 2:

- Initialize: $A = \phi$.
- **Do *recursion(1).***

Step 3:

- **If** $n \notin A$
  Do *recursion(n).*

Recursion(newnode)

- Initialize: $PT = \{newnode\}$; $DC = DC_{newnode}$; $A = A \cup \{newnode\}$.
- **Do** for each successor i of *newnode* which is not in A
  - ***Recursion(i)*** $\rightarrow PT', DC'$
  - **If** $DC' \geq 0$
    Merge: $PT = PT \cup PT'$; $DC = DC + DC'$

Else

Update current tree:

Delete *arc(newnode,i)*
Find new arc with minimal displacement
If an arc exists, add new arc
If no arc can be found, shift till deadline
Update completion times of nodes in *PT*

Repeat step 2 or 3

- **Do** for each predecessor *i* of *newnode* which is not in *A*:
  - ***Recursion(i)*** $\rightarrow PT', DC'$
  - *Merge*: $PT = PT \cup PT'$; $DC = DC + DC'$
- **Return**.

Given the structure of the algorithm, it is clear that only partial trees, consisting of two or more activities, can be the subject of a forward shift during the recursion. When the recursion checks for a predecessor of *newnode*, it is known that the corresponding partial tree must have a negative discounted cash flow and therefore, in this case only a merge is performed.

The procedure has been coded in Visual C++ for use on a personal computer and was validated against Grinold's procedure (Grinold 1972), adapted for AoN networks. Extensive computational tests obtained on a Digital Pentium 60 MHz Venturis machine on two data sets (98 test problems adapted from the 110 Patterson problem set (Patterson 1984) and 1980 networks adapted from the De Reyck and Herroelen set of ALB test problems (De Reyck and Herroelen 1996b)) reveal that the recursive search algorithm is very efficient. It finds the optimal solution for 1000 problem replications in an average of 0.433 seconds for the Patterson set and 0.420 seconds for the ALB problem set. It outperforms Grinold's procedure in that it is on the average 2.5 times faster on the Patterson set and 2.59 times faster on the ALB set at a much smaller CPU time variance.

De Reyck and Herroelen (1996d) have recently extended the Demeulemeester et al. (1996b) procedure to cope with the above discussed generalized precedence relations, which introduce arbitrary minimal and maximal time lags between the start and completion of activities. The procedure has been programmed in Microsoft® Visual C++2.0 under Windows NT for use on a Digital Venturis Pentium-60 personal computer. For the set of 7200 problem instances generated for the RCPSP-GPR (De Reyck and Herroelen 1996b) by ignoring the resource requirements and using uniformly generated cash flows in the interval [-500,+500], the required CPU times are very small (average computation times are smaller than 1 second, even for the 100-activity projects). The number of activities has a strong impact on the required computation time. Moreover, there is a positive correlation between the order strength (OS) and the required CPU time: when OS increases, the problem becomes harder.

5.2 The RCPSP with discounted cash flows (RCPSP-DC)

Adding renewable resource constraints to the model of Eqs. 27-39 yields the NP-hard (Baroum 1992) *resource-constrained project scheduling problem with discounted cash flows*. Icmeli and Erengüç (1996) present a branch-and-bound procedure for the resource-constrained max-npv problem. The project due date T is obtained as $T=s*D$, where D is the project duration obtained from a heuristic

solution procedure, and s is a constant greater than 1. The branch-and-bound procedure is to be considered an extension of the *DH-procedure* (Demeulemeester and Herroelen 1992) for solving the resource-constrained min-duration problem. At each node of the search tree a complete schedule which may be resource infeasible is obtained. At the initial node of the tree an optimal solution to the corresponding unconstrained max-npv problem is obtained using the fixed deadline algorithm of Grinold (1972), yielding an upper bound. If this solution is resource feasible the procedure terminates. If not, branching is done using the modifed version of the delaying scheme used by Demeulemeester and Herroelen (1992) to resolve resource conflicts, as described in the previous section. This modification is necessary because semi-active timetabling which starts activities as early as possible within the given constraints is inappropriate under the irregular npv-objective.

The subproblems generated by the branching process are solved using Grinold (1972). A node is fathomed either if the optimum unconstrained solution has a project duration exceeding the due date, or if it is less than or equal to that of the incumbent solution. The node with the greatest objective function value is selected for further branching.

The algorithm is written in Fortran and run on an IBM3090 computer with vector processing. In implementing the algorithm, the authors adopted tolerance levels which guarantee that the solution value obtained by the algorithm is within $(100\epsilon)\%$ of that of the optimal solution, with ϵ ranging from 0 to 0.05. The computational experiment used 50 problems taken from Patterson (1984) with cash flows generated randomly from a uniform distribution on [-500,1000], and 40 problems (32 activities, 3 resource types) generated using the *ProGen* generator (Kolisch et al. 1995) with cash flows generated from the uniform distribution on [-5000,10000]. Using 0% tolerance, 34 problems could be solved with a CPU time limit of 600 seconds (average CPU time ranging between 0.011 and 313 seconds) and a limit on the number of subproblems set to 4000. With the tolerance level increased to 0.05, only 9 problems (all in the ProGen set) remained unsolved. The algorithm was also shown to outperform the procedure by Yang et al. (1992), which could only solve 10 problems, exceeding the CPU time limit for the remaining 80 problems.

It is to be expected that computational gains can be obtained from solving the subproblems using the optimal recursive search algorithm of Demeulemeester et al. (1996b) for solving the max-npv problem instead of Grinold's procedure.

6. Minimizing resource availability costs

Basically, the procedures discussed so far, provide an answer to the following question: *Given the project data and the resource availabilities, what is the shortest project length that can be obtained such that no precedence or resource constraints are violated ?* Given a project due date T and a discrete, non-decreasing cost function $c_k(a_k)$ of the constant resource availability a_k of renewable resource type k, the resource availability cost problem (RACP) aims at determining the cheapest resource availability amounts for which a feasible schedule exists that does not violate the project due date. Conceptually, the RACP can be formulated as follows:

$$\min \sum_{k=1}^{K} c_k(a_k) \qquad [30]$$

subject to

$$f_i \leq f_j - d_j \qquad \text{for all } (i,j) \in H \qquad [31]$$

$$f_1 = 0 \qquad [32]$$

$$f_n = T \qquad [33]$$

$$\sum_{i \in S_t} r_{ik} \leq a_k \qquad k = 1,2,\ldots,K; t = 1,2,\ldots,f_n \qquad [34]$$

Eq. 33 specifies the project due date. The remaining equations correspond to Eqs. 2-4 of the RCPSP. The decision variables, however, are the renewable resource availabilities a_k and the finish times f_i.

Demeulemeester (1995) has developed an optimal solution procedure based on a search strategy which starts by determining the minimum resource availabilities required for the different resource types. These are derived from the solution of resource-constrained project scheduling decision problems, either with a single resource type or with two resource types (for all other resource types the availability is assumed to be infinite). Based on the solutions to these problems (obtained by the DH-procedure and its extensions), the algorithm defines so-called *efficient points*, which delimit the solution space of all possible combinations of resource availabilities that are not eliminated by solving the resource-constrained project scheduling decision problems with one or two resource types (point i with resource availabilities a_1, ..., a_K is an efficient point if no other point j with resource availabilities $a'_1,\ldots,a'_K$ exists in the solution space such that all $a'_k \leq a_k$ for $k=1,\ldots,K$). One then tries to solve the resource-constrained project scheduling decision problem that corresponds with the cheapest efficient point (which also is the cheapest point of the entire solution space as no point in the solution space has a resource availability cost that is smaller than the cost of the cheapest efficient point). If no feasible solution can be found with these resource availabilities, the efficient point is cut from the solution space, new efficient points are defined for which the resource availability for one resource type is one unit higher and the search is continued by trying to solve the resource-constrained project scheduling problem that corresponds with the currently cheapest efficient point. This process is repeated until a feasible solution is found. The resource availabilities that correspond with the efficient point for which the first feasible solution was found constitute the optimal solution.

The computational experience obtained with the proposed algorithm shows it to outperform Möhring's procedure (Möhring 1984), which is the only solution procedure available for optimally solving the minimal resource availability cost problem. Moreover, the procedure proves to be less sensitive to changes in the cost parameters. Computational experience on an adapted Patterson problem set reveals that the procedure needs an average computation time of about 0.55 seconds on an IBM PS/2 with a 486 processor running at 25 MHz in order to obtain a 5% reduction in the resource availability cost with respect to the optimal RCPSP schedule. The average computation time as well as the standard deviation of the computation time are increasing functions of the number of resource types.

Projects with up to 6 resource types seem to be the maximum with which the procedure can effectively cope.

7. Discrete trade-offs in project scheduling

Various types of trade-offs have been studied in the context of project scheduling. In this section we focus on discrete time/cost trade-offs, discrete time/resource and resource-resource trade-offs.

7.1 The discrete time/cost trade-off problem (DTCTP)

Demeulemeester et al. (1995) have developed two optimal procedures for the discrete time/cost trade-off problem (DTCTP) in deterministic project networks of the CPM type, under a single *nonrenewable* resource. The specification of a project is assumed to be given in activity-on-arc (AoA) notation by a directed acyclic graph (dag) $D = (N,A)$ in which N is the set of nodes, representing network "events", and A is the set of arcs, representing network "activities". It is assumed, without loss of generality, that there is a single start node *1* and a single terminal node n, $n=|N|$. The duration y_a of activity $a \in A$ is a discrete, nonincreasing function $g_a(x_a)$ of the amount of a single resource allocated to it; i.e., $y_a = g_a(x_a)$. The pair y_a,x_a shall be referred to as a *"mode"*, and shall be written as: $y_a(x_a)$. Thus an activity that assumes four different durations according to four possible resource allocations to it shall be said to possess four modes.

Three possible objective functions for the DTCTP (see also De et al. 1995) are considered. For the *first objective function* (referred to as *P1*) we specify a limit R on the total availability of a single nonrenewable resource type. The problem is then to decide on the vector of activity durations $(y_1,...,y_m)$, $m = |A|$, that completes the project as early as possible under the limited availability of the single nonrenewable resource type. A *second objective function* (referred to as *P2*) reverses this problem formulation: now we specify a limit T on the project length and we try to minimize the sum of the resource usage over all activities. For the third and *final objective function* (referred to as *P3*) we have to compute the complete time/cost trade-off function for the total project, i.e., in the case of the DTCTP all the efficient points (T,R) such that with a resource limit R a project length T can be obtained and such that no other point (T', R') exists for which both T' and R' are smaller than or equal to T and R:

$$\min \left[\sum_{(ij)\in A} x_{ij} \;\middle|\; t_{-n} \leq T \leq \bar{t}_n \right]$$

The early contributions to the basic time-cost trade-off problem in CPM networks assumed ample resource availability and tried to minimize the total project cost subject to precedence constraints and lower and upper bound constraints on the activity durations. While the problem has been widely studied under the assumption of continuous time-cost relationships (see standard texts such as Moder et al. 1983), the literature on the DTCTP where the time-cost relationships are defined at discrete points has

been rather sparse. De et al. (1995) offer an excellent review and have shown (De et al. 1992) that any exact solution algorithm would very likely exhibit an exponential worst-case complexity.

The two optimal procedures developed by Demeulemeester et al. (1995) are based on dynamic programming logic. The first procedure (*Reduction Plan 1*) heavily relies on a network reduction scheme proposed by Bein et al. (1992) and subsequently referred to as the *BKS approach*. Series-parallel networks can be reduced by a cascade of *series-parallel reductions* through the application of dynamic programming. The serial optimization goes as follows: if $a = (i,j)$ is the unique arc into j and $b = (j,k)$ is the unique arc out of j then these two arcs in series are replaced by a single arc $c = (i,k)$. The parallel optimization process can be viewed as parallel arc reduction: two or more parallel arcs $a_1, \ldots, a_m$ leading from i to j are replaced by a unique arc $a = (i,j)$. A project that can be optimized through a succession of series and parallel optimizations (reductions) is said to be *series/parallel reducible* (*s/p reducible*, for short). A project which cannot be thus optimized is called *s/p irreducible*. An efficient method for s/p irreducible networks consists of enumerating the cost assignments for a limited number of activities, preferably the minimum number, such that the resulting network becomes s/p reducible. We refer to this method as "optimal fixing". A *"reduction plan"* now consists of all actions that have to be performed on a network in order to reduce the network to one single arc, including the determination of which activities need to be fixed. As soon as such a plan is constructed it is quite easy to obtain a solution to all of the objective functions that we proposed.

The BKS approach is composed of two major steps: the first constructs the "complexity graph" of the given project network (easily accomplished from standard "dominator tree" arguments); and the second determines the minimal node cover of this complexity graph. The key element in their construction is that the complexity graph is *directed and acyclic*, whence its minimal node cover can be easily secured by a simple (i.e., polynomially bounded) "maximum flow" procedure.

The BKS scheme yields the *minimum number of node reductions*. Such a scheme, however, may not minimize the computational effort required in reducing the network because we should consider not only the *number of node reductions* but also the *order* in which the reductions are performed (often there is a choice), as well as the *total number of leaves* which have to be evaluated. These additional factors may have a dominant effect on the computation time, as evidenced by our experimentation. In the enumerative scheme adopted (*Reduction Plan 2*), Demeulemeester et al. (1995) have termed a complete set of resource allocations to all the "index activities" in the project a *"leaf"*. They developed a branch-and-bound procedure for generating a reduction plan that aims at minimizing the total number of leaves evaluated, which is equivalent to minimizing the computing effort.

The suggested procedures were programmed for personal computer. In the absence of a standard set of test problems for the DTCTP, the procedure was extensively tested through Monte Carlo experimentation on a variety of networks drawn from the literature or generated specifically for this study. The results are most encouraging: for projects up to 20 nodes and 45 activities (with the number of execution modes for each activity generated from a discrete uniform distribution in the range [1,10]), the time required never exceeded 7 minutes. Furthermore, the authors recommend the use of Plan 2 for

projects with a large number of activities and high complexity index. It outperforms Plan 1 both in terms of CPU time required and number of leaves visited. The experiments also confirmed the dominant role played by the complexity index, CI, and suggested its use as a measure of network complexity.

7.2 The discrete time/resource trade-off problem (DTRTP)

In the RCPSP each activity has a *single* execution *mode*: both the activity duration and its requirements for a set of renewable resources are assumed to be fixed. Herroelen (1968) and Elmaghraby (1977) were the first authors to deal with discrete time-resource trade-offs and, correspondingly, multiple ways for executing the project activities.

In practice, it often occurs that only one (renewable) resource is present (e.g. labor), and that for each activity a work content (e.g. man-days) is specified instead of a set of activity modes with preset durations and corresponding resource requirements. In other words, each activity mode with a work content at least as high as the specified activity work content is allowed. In the *discrete time/resource trade-off problem (DTRTP)*, the duration of an activity is a discrete, non-increasing function of the amount of a single *renewable* resource committed to it. Given the specified work content W_i (e.g. man-days) for activity i, $1 \le i \le n$, all M_i efficient execution modes for its execution are determined based on time/resource trade-offs. Activity i when performed in mode m, $1 \le m \le M_i$, has a duration d_{im} and requires a constant amount r_{im} of the renewable resource, during each period it is in progress, such that $r_{im}d_{im}$ is at least equal to and as close as possible to W_i. A mode is called efficient if there is no other mode with equal duration and lower resource requirement or equal resource requirement and smaller duration. Without loss of generality, it is assumed that the modes of each activity are sorted in the order of non-decreasing duration. The single renewable resource has a constant per period availability a. We assume that the dummy start node *1* and the dummy end node n have a single execution mode with zero duration and zero resource requirement. The objective is to schedule each activity in one of its execution modes, subject to the finish-start precedence and the renewable resource constraint, under the objective of minimizing the project makespan. Introducing the decision variables

$$x_{imt} = \begin{cases} 1, & \text{if activity } i \text{ is performed in mode } m \text{ and started at time } t \\ 0, & \text{otherwise} \end{cases}$$

the DTRTP can be formulated as follows:

$$\text{Minimize} \sum_{t=e_n}^{l_n} t\, x_{nlt} \qquad [35]$$

Subject to

$$\sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} x_{imt} = 1 \qquad i = 1,2,\ldots,n \qquad [36]$$

$$\sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} \left(t + d_{im}\right) x_{imt} \le \sum_{m=1}^{M_j} \sum_{t=e_j}^{l_j} t\, x_{jmt} \qquad (i,j) \in E \qquad [37]$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{im} \sum_{s=\max\{t-d_{im},\, e_i\}}^{\min\{t-1,\, l_i\}} x_{ims} \le a \qquad t = 1,2,\ldots,T \qquad [38]$$

$$x_{imt} \in \{0,1\} \qquad i = 1,2,\ldots,n \quad m = 1,2,\ldots,M_i \quad t = 0,1,\ldots,T \qquad [39]$$

with

$e_i\,(l_i)$ = critical path based earliest (latest) start time of activity i based on the modes with the smallest duration

T = upper bound on the project duration

E = set of precedence related activities

The objective function Eq. 35 minimizes the makespan of the project. Constraint set Eqs. 36 ensures that each activity is assigned exactly one mode and exactly one start time. Constraints Eqs. 37 denote the precedence constraints. Constraints Eqs. 38 secure that the per-period availability of the renewable resource is met. Finally, constraints Eqs. 39 force the decision variables to assume 0-1 values.

The DTRTP resembles the *discrete time/cost trade-off problem (DTCTP)*, discussed above, which studies time/cost trade-offs for a single *nonrenewable* resource. Instead of dealing with a single nonrenewable resource, the DTRTP is concerned with a single renewable resource. Also, the DTRTP is a subproblem of the *multi-mode resource-constrained project scheduling problem (MRCPSP)*, which includes time/resource and resource/resource trade-offs, multiple renewable, nonrenewable and doubly-constrained resources (limited on a per period basis and a total project basis) and a variety of objective functions (Sprecher and Drexl 1996a,b). As a generalization of the RCPSP, the DTRTP is NP-hard.

Demeulemeester et al. (1996c) present two optimal procedures for solving the DTRTP: a mode generation procedure and an integrated approach. The mode generation procedure enumerates all possible *mode combinations* for the network (a mode combination assigns each activity one of its possible modes). For each mode combination, the resulting RCPSP is solved by running the DH1-procedure developed by Demeulemeester and Herroelen (1996d). At the start of each run, the upper bound T is taken as the minimum makespan of the feasible schedules obtained in the previous runs. At each decision point (corresponding to the completion times of one or more activities), the integrated depth-first branch-and-bound procedure evaluates the feasible partial schedules (nodes in the search tree) obtained by enumerating all *feasible maximal activity-mode combinations*. Each maximal activity-mode combination is evaluated by computing a critical path-based and a resource-based lower bound. Backtracking occurs when a schedule is completed or when a branch is to be fathomed by the lower bound calculation or one of the dominance rules. The procedure stops with the verified optimal solution

upon backtracking to level 0 of the search tree. The mode generation procedure uses bounds and dominance rules of the DH1-procedure. The new integrated procedure applies precedence and resource-based bounds in combination with various dominance rules (a cutset dominance rule, a single-mode left-shift rule, a multi-mode left-shift rule - other mode, earlier completion time - and a mode reduction rule - shorter mode, same completion time - and some specific dominance criteria).

Research is underway to validate the procedures. It is hoped that the integrated procedure yields promising results and holds the potential of extension to the multi-mode case.

7.3 The multi-mode case

As mentioned above, the multi-mode resource-constrained project scheduling problem (MRCPSP) includes time/resource and resource/resource trade-offs, multiple renewable (limited on a per period basis), nonrenewable (limited for the entire project) and doubly-constrained resources (limited on both a per period and total project basis) and a variety of objective functions. In the basic problem setting, activities have to be scheduled in one of their possible execution modes subject to renewable and nonrenewable resource constraints in order to minimize the project duration. Doubly-constrained resources can easily be taken into account by enlarging the sets of renewable and nonrenewable resources. Usually the execution modes for an activity are numbered in increasing order of the activity duration.

Sprecher et al. (1994) have extended the DH-procedure to the multi-mode case under the minimum makespan objective. They borrow the notion of tight schedules from Speranza and Vercellis (1993) and introduce the notion of mode-minimal schedules. A schedule is tight if there does not exist an activity the finish time of which can be reduced without violating the constraints or changing the finish time or mode of any of the remaining activities in progress. Tight schedules can be transformed into mode-minimal schedules by applying mode reduction (same finishing time, other mode; the modes are labeled with respect to non-decreasing duration). They show that if there is an optimal schedule for a given instance, then there is an optimal schedule which is both tight and mode-minimal. Sprecher et al. (1994) use a branching scheme wich fixes the mode of eligible activities before putting them in progress. Resource conflicts are then resolved through the logic of the DH-procedure; i.e., by delaying minimal delaying alternatives. The algorithm has been coded in Borland C for an IBM-compatible 386-DX personal computer with 40 MHz clockpulse, and has been tested on 536 instances generated using *ProGen*. Each instance consists of 12 activities (including dummy start and end nodes), three possible execution modes with duration varying between 1 and 10 periods, two renewable and two nonrenewable resources. The problems are solved in an average CPU time of 0.53 seconds. As such it outperforms previously developed procedures by Sprecher (1994) and Speranza and Vercellis (1993) (it has been shown by Hartmann and Sprecher (1993) that this procedure may miss the optimum).

Sprecher and Drexl (1996a,b) have subsequently developed a branch-and-bound procedure which relies on an enumeration scheme based on the precedence tree concept introduced by Patterson et al. (1989,1990) and already used by Sprecher (1994). In the precedence tree, an activity is considered to become eligible (and to become a descendant of a parent node in the search tree) if all its predecessors

are scheduled but not necessarily finished. The basic scheme is enhanced by different static and dynamic search tree reduction schemes, preprocessing and bounding rules. The procedure has been coded in GNU C and runs under OS/2 on a personal computer (80486dx processor, 66MHz clockpulse, 16 Mb memory). More than ten thousand problem instances have been generated using ProGen to evaluate the algorithm's performance. The number of activities in the test instances ranges from 10 to 20, from 1 up to 5 execution modes, 1 up to 5 renewable and 1 up to 3 nonrenewable resources. For the basic problem subset used to evaluate ProGen (Kolisch et al. 1995); i.e. 536 ten-activity problems, 3 modes, 2 renewable and 2 nonrenewable resources, the authors report an average CPU time of 0.14 seconds (standard deviation of 0.21 seconds, with a maximum of 2.31 seconds). Over the complete set of instances, CPU times seem to increase exponentially with the number of jobs and modes and seems to decrease with an increasing complexity (measured by the above mentioned CNC). The number of renewable resources seems to influence the CPU times linearly. The number of nonrenewable resources have a strong (positive correlation) impact on CPU times. The higher the resource factor RF, and the lower the resource strength RS, the higher the CPU time required. Encouraging results are reported using a truncated version of the algorithm. The Sprecher and Drexl (1996a,b) procedure clearly constitutes a benchmark for the MRCPSP.

8. Conclusions

Over the past decade, and especially over the past five years, considerable progress has been made in designing optimal solution procedures for the resource-constrained project scheduling problem (RCPSP). While at the time of the first extensive performance evaluation of optimal enumeration procedures (Patterson 1984), only one procedure (Stinson et al. 1978) was capable of solving all the 110 Patterson test problems on a mainframe, we now witness the situation that all problems can be solved optimally by the DH2 procedure in an average CPU time of 0.002 seconds on a Pentium Pro processor with 200 MHz clock pulse. These remarkable results made it happen that the 110 Patterson problem set, an assembled set of test problems (7-50 activities, 1-3 renewable resource types) which do not satisfy preset values of problem parameters and which for many years has served as the de facto standard test set, can no longer uniquely serve as the benchmark test set for the RCPSP.

New optimal procedures (and definitely, new suboptimal procedures) should be validated on a wider set of test instances, generated to satisfy preset values of relevant problem parameters. ProGen, the problem generator developed by Kolisch et al. (1995), has been used to generate a set of 480 RCPSP test instances (32 activities, 4 renewable resource types) which currently serves as the challenging test set. The DH2 procedure has recently optimally solved all problems in this set in an average CPU time of 0.372 seconds (standard deviation 2.744 seconds with a maximum of 50.97 seconds) on the 200 MHz Pentium Pro.

Clearly, the state of the art is such that properly designed depth-first branch-and-bound procedures offer the best potential for solving the RCPSP. The underlying logic (branching strategy, dominance and bounding rules) of the DH-procedure (Demeulemeester and Herroelen 1992) and its extensions DH1 (Demeulemeester and Herroelen 1996d) and DH2 has a wide field of application as witnessed by

the possible extensions into a number of important derived resource-constrained scheduling problems (see Table I).

Table I. Extensions of DH solution concepts

	RCPSP	PRCPSP	GRCPSP	RCPSP-GPR	DTRTP	MRCPSP	RCPSP-DC
semi-active timetabling	yes	yes	yes	no	yes	yes	no
min delaying alternatives (or max scheduling alternatives)	yes	yes	yes	yes	yes	yes	yes
delaying alternatives	yes	yes	yes	no	yes	yes	no
delaying modes	no	no	no	yes	no	no	yes
LB0	yes	yes	yes	yes	yes	yes	no
LB1	yes	yes	no	no	yes	yes	no
LB2	yes	yes	no	no	yes	yes	no
LB3	yes	yes	yes	yes	yes	yes	no
Theorem 1	yes	yes	yes	no	yes	yes	no
Theorem 2	yes	yes	yes	no	yes	yes	no
Theorem 3	yes	yes	yes	no	yes	yes	no
Theorem 4	yes	yes	yes	no	yes	yes	no
subset dominance	yes	yes	yes	yes	yes	yes	yes

The branching scheme which resolves resource conflicts by delaying minimal delaying alternatives (or an equivalent scheme which starts maximal scheduling alternatives) has proven to be very robust and efficient in a wide variety of problem settings. The various studies confirm the importance of easy-to compute and sufficiently strong bounding and dominance rules. Especially the cutset dominance rule has proven its effectiveness, especially when sufficient memory is available and efficient data structures are used for storing the cutsets. Important trade-offs do exist between the strength of a bounding rule and the computational effort required for its computation. As revealed by the experience gained with LB3 and its variations, the balance involved can be very delicate. Research efforts aimed at further exploiting this delicate balance is more than justified.

The various validation experiments performed on optimal solution procedures for the RCPSP and its problem variants clearly reveal the importance of intelligent computer coding. Investing in the development of effective and efficient data structures and fully exploiting the full potential of existing computer platforms and 32-bit programming clearly pays off.

Computational experience gained on a wide variety of test instances confirms the rich potential of truncated branch-and-bound procedures. Often optimal solutions have been found within short amounts of computation time. Often also, the quality of the solutions obtained by truncated branch-and-bound procedures competes, if not outperforms, the solution quality of many heuristics. Truncated exact procedures are promising tools for solving real problems (of sufficiently large size) within an acceptable computational burden and with acceptable solution quality.

Most computational experience has been gained on the 110 Patterson problem set and the test problems generated using ProGen (a project scheduling problem library has been made available on a ftp-site by Kolisch and Sprecher (1996)). Results confirm that the 110 test problems no longer serve as the de facto standard. Gaining computational experience on a set of test problems which satisfy preset values of relevant problem parameters and which span the full range of complexity is crucial. This

warrants further research on the issue of establishing additional relevant factors which explain the real complexity of a problem instance. Efforts to incorporate such parameters in the existing problem generators, or removing others with almost no explanatory power, should make it possible to establish workable problem test sets as a base for full factorial experiments which can be used for validating optimal and suboptimal procedures for solving the RCPSP and its important, realistic problem derivates.

References

Alvarez-Valdes, R. and J.M. Tamarit (1989), Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and Empirical Analysis, in R. Slowinski and J. Weglarz (eds.), *Advances in Project Scheduling*, Elsevier, Amsterdam.

Ashour, S. (1972), *Sequencing Theory*, Springer, Berlin.

Baker, K.R. (1974), *Introduction to Sequencing and Scheduling*, Wiley.

Balas, E. (1971), Project Scheduling with Resource Constraints. In Beale, E.M.L. (editor), *Applications of Mathematical Programming*, The English University Press, London, 187-200.

Baroum, S.M. (1992), An Exact Solution Procedure for Maximizing the Net Present Value of Resource-Constrained Projects, unpublished Ph.D. dissertation, Indiana University.

Bartusch, M., R.H. Möhring and F.J. Radermacher (1988), Scheduling Project Networks with Resource Constraints and Time Windows, *Annals of Operations Research*, 16, 201-240.

Bein, W.W., J. Kamburowski and M.F.M. Stallmann (1992), Optimal Reduction of Two-Terminal Directed Acyclic Graphs, *SIAM Journal on Computing*, 21, 1112-1129.

Bell, C.A. and K. Park (1990), Solving Resource-Constrained Project Scheduling Problems by A^* Search, *Naval Research Logistics*, 37, 61-84.

Bellmann, R., A.O. Esogbue and I. Nabeshima (1982), *Mathematical Aspects of Scheduling and Applications*, Pergamon Press.

Blazewicz, J., J.K. Lenstra and A.H.G. Rinnooy Kan (1983), Scheduling Projects to Resource Constraints: Classification and Complexity, *Discrete Applied Mathematics*, 5, 11-24.

Blazewicz, J., K. Ecker, G. Schmidt and J. Weglarz (1993), *Scheduling in Computer and Manufacturing Systems,* Springer, Berlin.

Bowman, E.H. (1959), The Schedule-Sequencing Problem, *Operations Research*, 7, 621-624.

Brand, J.D., W.L. Meyer and L.R. Shaffer (1964), The Resource Scheduling Problem in Construction, Civil Engineering Studies, Report N° 5, Department of Civil Engineering, University of Illinois, Urbana, Ill.

Brinkmann, K. and K. Neumann (1994), Heuristic Procedures for Resource-Constrained Project Scheduling with Minimal and Maximal Time Lags: The Minimum Project Duration and Resource Levelling Problem, Technical Report WIOR-443, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.

Brucker, P. (1995), *Scheduling Algorithms*, Springer, Berlin.

Brucker, P., A. Schoo and O. Thiele (1996), A Branch-and-Bound Algorithm for the Resource-Constrained Project Scheduling Problem, Preprint Reihe P, Heft 178, Osnabrücker Schriften zur Mathematik, Universität Osnabrück.

Carlier, J. and B. Latapie (1991), Une méthode arborescente pour les problèmes cumulatifs, *R.A.I.R.O.*, 25, 311-340.

Carlier, J. and E. Neron (1996), A New Branch and Bound Method for Solving the Resource Constrained Project Scheduling Problem, *PMS'96, The Fifth International Workshop on Project Management and Scheduling*, Poznan, April 11-13, 61-65.

Carruthers, J.A. and A. Battersby (1966), Advances in Critical Path Methods, *Operational Research Quarterly*, 17, 359-380.

Christofides, N., R. Alvarez-Valdes and J.M. Tamarit (1987), Project Scheduling with Resource Constraints: A Branch and Bound Approach, *European Journal of Operational Research*, 29, 262-273.

Conway, R.W., W.L. Maxwell and L.W. Miller (1967), *Theory of Scheduling*, Addison-Wesley Publishing Company.

Cooper, D.F. (1976), Heuristics for Scheduling Resource-Constrained Projects: An Experimental Comparison, *Management Science*, 22, 1186-1194.

Dar-El, E.M. (1973), MALB - A Heuristic Technique for Balancing Large Single-Model Assembly Lines, *AIIE Transactions*, 5, 343-356.

Davis, E.W. (1966), Resource Allocation in Project Network Models - A Survey, *Journal of Industrial Engineering*, 17, 177-188.

Davis, E.W. (1973), Project Scheduling Under Resource Constraints: Historical Review and Categorization of Procedures, *AIIE Transactions*, 5, 297-313.

Davis, E.W. (1975), Project Network Summary Measures and Constrained Resource Scheduling, *IIE Transactions*,7, 132-142.

Davis, E.W. and G.E. Heidorn (1971), An Algorithm for Optimal Project Scheduling Under Multiple Resource Constraints, *Management Science*, 27, B803-B816.

De, P., E.J. Dunne, J.B. Gosh and C.E. Wells (1992), Complexity of the Discrete Time-Cost Tradeoff Problem for Project Networks, Tech. Report, Dept. MIS and Dec. Sci., University of Dayton, Dayton, OH.

De, P., E.J. Dunne, J.B. Gosh and C.E. Wells (1995), The Discrete Time-Cost Tradeoff Problem Revisited, *European Journal of Operational Research*, 81, 225-238.

Deckro, R.F., E.P. Winkofsky, J.E. Hebert and R. Gagnon (1991), A Decomposition Approach to Multi-Project Scheduling, *European Journal of Operational Research*, 51, 110-118.

Demeulemeester, E. (1992), Optimal Algorithms for Various Classes of Multiple Resource-Constrained Project Scheduling Problems, Ph.D. Thesis, Department of Applied Economic Sciences, Katholieke Universiteit Leuven.

Demeulemeester, E. (1995), Minimizing Resource Availability Costs in Time-Limited Project Networks, *Management Science*, 41, 1590-1598.

Demeulemeester, E. and W. Herroelen (1992), A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem, *Management Science*, 38, 1803-1818.

Demeulemeester, E. and W. Herroelen (1996a), A Branch-and-Bound Procedure for the Generalized Resource-Constrained Project Scheduling Problem, *Operations Research*, to appear.

Demeulemeester, E. and W. Herroelen (1996b), An Efficient Optimal Solution Procedure for the Preemptive Resource-Constrained Project Scheduling Problem, *European Journal of Operational Research*, 90, 334-348.

Demeulemeester, E.L. and W.S. Herroelen (1996c), Modelling Setup Times, Process Batches and Transfer Batches Using Activity Network Logic, *European Journal of Operational Research*,.89, 355-365.

Demeulemeester, E.L. and W.S. Herroelen (1996d), New Benchmark Results for the Resource-Constrained Project Scheduling Problem, *Management Science*, to appear.

Demeulemeester, E., B. De Reyck and W. Herroelen (1996a), Optimal and Suboptimal Procedures for the Discrete Time/Resource Trade-Off Problem in Project Networks, *PMS'96 - The Fifth International Workshop on Project Management and Scheduling*, April 11-13, Poznan, 84-87.

Demeulemeester, E.L., W.S. Herroelen and S.E. Elmaghraby (1995), Optimal Procedures for the Discrete Time/Cost Trade-Off Problem in Project Networks, *European Journal of Operational Research*, 88, 50-68.

Demeulemeester, E., W. Herroelen, W.P. Simpson, S. Baroum, J.H. Patterson and K.-K. Yang (1994), On a Paper by Christofides et al. for Solving the Multiple-Resource Constrained Single Project Scheduling Problem, *European Journal of Operational Research*, 76, 218-228.

Demeulemeester, E., W. Herroelen and P. Van Dommelen (1996b), An Optimal Recursive Search Procedure for the Deterministic Unconstrained Max-npv Scheduling Problem, Research Report 9603, Department of Applied Economics, K.U.Leuven.

De Reyck, B. (1995a), Project Scheduling Under Generalized Precedence Relations - A Review: Part 1 and 2, Research Reports 9517/18, Department of Applied Economics, K.U.Leuven.

De Reyck, B. (1995b), On the Use of the Restrictiveness as a Measure of Complexity for Resource-Constrained Project Scheduling, Research Report 9535, Department of Applied Economics, K.U.Leuven, Belgium.

De Reyck, B. and W. Herroelen (1995), Assembly Line Balancing by Resource-Constrained Project Scheduling Techniques - A Critical Appraisal,.Research Report 9505, Department of Applied Economics, K.U.Leuven, Belgium.

De Reyck, B. and W. Herroelen (1996a), On the Use of the Complexity Index as a Measure of Complexity in Activity Networks, *European Journal of Operational Research*, 91, 347-366.

De Reyck, B. and W. Herroelen (1996b), A Branch-and-Bound procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Constraints, Research Report N° 9613, Department of Applied Economics, K.U.Leuven.

De Reyck, B. and W. Herroelen (1996c), Computational Experience with a Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations, Research Report 9628, Department of Applied Economics, K.U.Leuven.

De Reyck, B. and W. Herroelen (1996d), An Optimal Procedure for the Unconstrained Max-NPV Project Scheduling Problem with Generalized Precedence Relations, Research Report 9642, Department of Applied Economics, K.U.Leuven, Belgium.

Elmaghraby, S.E. (1967), The Sequencing of N Jobs on M Parallel Processors, Unpublished paper, North Carolina State University at Raleigh, Raleigh, U.S.A.

Elmaghraby, S.E. (1977), *Activity Networks: Project Planning and Control by Network Models*, John Wiley and Sons, New York.

Elmaghraby, S.E. (1995), Activity Nets: A Guided Tour Through Some Recent Developments, *European Journal of Operational Research*, 82, 383-408.

Elmaghraby, S.E. and W.S. Herroelen (1980), On the Measurement of Complexity in Activity Networks, *European Journal of Operational Research*, 5, 223-234.

Elmaghraby, S.E. and W.S. Herroelen (1990), The Scheduling Activities to Maximize the Net Present Value of Projects, *European Journal of Operational Research*, 49, 35-49.

Elmaghraby, S.E. and J. Kamburowski (1992), The Analysis of Activity Networks Under Generalized Precedence Relations, *Management Science*, 38, 1245-1263.

Franck, B. and K. Neumann (1996), Priority-Rule Methods for the Resource-Constrained Project Scheduling Problem with Minimal and Maximal Time Lags - An Empirical Analysis, *PMS'96 - Fifth International Workshop on Project Management and Scheduling*, April 11-13, Poznan, 88-91.

French, S. (1982), *Sequencing and Scheduling - An Introduction to the Mathematics of the Job-Shop*, Wiley.

Gargeya, V.B. and R.H. Deane (1996), Scheduling Research in Multiple Resource Constrained Job Shops: A Review and Critique, *International Journal of Production Research*, 34, 2077-2097.

Grinold, R.C. (1972), The Payment Scheduling problem, *Naval Research Logistics Quarterly*, 19, 123-136.

Hartmann, S. and A. Sprecher (1993), A Note on "Hierarchical Models for Multi-Project Planning and Scheduling", Research Report N° 338, Christian-Albrechts-Universität zu Kiel.

Herroelen, W.S. (1968), *Een probleem van "resource allocation": het toewijzen van netwerkactiviteiten aan ingenieurs*, comm. eng. thesis, Dept. Appl. Econ., K.U. Leuven.

Herroelen, W.S. (1972), Resource-Constrained Project Scheduling - The State of the Art, *Operational Research Quarterly*, 23, 261-275.

Herroelen, W.S. (1991), *Operationele produktieplanning*, Acco, Leuven.

Herroelen, W.S. and E. Gallens (1993), Computational Experience with an Optimal Procedure for the Scheduling of Activities to Maximize the Net Present Value of Projects, *European Journal of Operational Research*, 65, 274-277.

Herroelen, W.S. and E.L. Demeulemeester (1994), Resource-Cconstrained Project Scheduling - A View on Recent Developments, *Tijdschrift voor Economie en Management, XXXIX, 371-395*

Herroelen, W.S. and E.L. Demeulemeester (1995), Recent Advances in Branch-and-Bound Procedures for Resource-Constrained Project Scheduling Problems, Chapter 12 in *Scheduling Theory and Its Applications* (co-editors Ph. Chrétienne, E.G. Coffmann Jr., J.K. Lenstra and Z. Liu), John Wiley and Sons, 259-276.

Herroelen, W.S., P. Van Dommelen and E.L. Demeulemeester (1996), Project Network Models with Discounted Cash Flows - A Guided Tour Through Recent Developments, *European Journal of Operational Research*, to appear.

Icmeli, O. and S. Erenguç (1996), A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Discounted Cash Flows, *Management Science*, to appear.

Icmeli, O., S.S. Erengüç and J.C. Zappe (1993), Project Scheduling Problems: A Survey, *International Journal of Production and Operations Management*, 13, 80-91.

Icmeli, O. and W.O. Rom (1996), Solving the Resource-Constrained Project Scheduling Problem with Optimization Subroutine Library, *Computers and Operations Research*, 23, 801-817.

Johnson, T.J.R. (1967), An Algorithm for the Resource Constrained Project Scheduling Problem, PhD Dissertation, MIT.

Kao, E.P.C. and M. Queyranne (1992), On Dynamic Programming Methods for Assembly Line Balancing, *Operations Research*, 30, 375-390.

Kaplan, L., (1988), Resource-Constrained Project Scheduling with Preemption of Jobs, Ph.D. dissertation, University of Michigan.

Kaplan, L. (1991), Resource-Constrained Project Scheduling with Setup Times, Unpublished paper,

Department of Management, University of Tenessee, Knxville.
Kerbosch, J.A.G.M. and H.J. Schell (1975), Network Planning by the Extended METRA Potential Method, Report KS-1.1, University of Technology Eindhoven, Department of Industrial Engineering.
Kolisch, R., A. Sprecher and A. Drexl (1995), Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems, *Management Science*, 41, 1693-1703.
Kolisch, R. and A. Sprecher (1996), PSPLIB - A Project Scheduling Problem Library, Research Report N° 396, Christian-Albrechts-Universität zu Kiel.
Kurtulus, I.S. and S.C. Narula (1985), Multi-Project Scheduling: Analysis of Project Performance, *IIE Transactions*, 17, 58-66.
Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (1993), Sequencing and Scheduling: Algorithms and Complexity, in Graves et al. (1993), *Handbooks in OR and MS - Vol. 4: Logistics of Production and Inventory*, North-Holland.
Mastor, A.A. (1970), An Experimental and Comparative Evaluation of Production Line Balancing Techniques, *Management Science*, 16, 728-746.
Mingozzi, A., V. Maniezzo, S. Ricciardelli and L. Bianco (1995), An Exact Algorithm for Project Scheduling with Resource Constraints Based on a New Mathematical Formulation, Revised Technical Report, University of Bologna, October 1995.
Moder, J.J., C.R. Phillips and E.W. Davis (1983), *Project Management with CPM, PERT and Precedence Diagramming*, Van Nostrand Reinhold.
Möhring, R.H. (1984), Minimizing Costs of Resource Requirements in Project Networks Subject to a Fixed Completion Time, *Operations Research*, 32, 89-120.
Möhring, R.H. and F.J. Radermacher (1989), The Order-Theoretic Aproach to Scheduling: the Deterministic Case, *Advances in Project Scheduling* (Slowinskli, R. and J. Weglarz (eds.)), Part I, Chapter 2, Elsevier.
Moodie, C.L. and D.E. Mandeville (1966), Project Resource Balancing by Assembly Line Balancing Techniques, *Journal of Industrial Engineering*, 17, 377-383.
Morton, Th.E. and D.W. Pentico (1993), *Heuristic Scheduling Systems - With Applications to Production Systems and Project Management*, Wiley Interscience.
Neumann, K. and C. Schwindt (1995), Projects with Minimal and Maximal Time Lags: Construction of Activity-on-Node Networks and Applications, Technical Report WIOR-447, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.
Neumann, K. and J. Zahn (1996), Heuristics for the Minimum Project-Duration Problem with Minimal and Maximal Time Lags Under Fixed Resource Constraints, *Journal of Intelligent Manufacturing*, to appear.
Özdamar, L. and G. Ulusoy (1995), A Survey on the Resource-Constrained Project Scheduling Problem, *IIE Transactions*, 27, 574-586.
Pascoe, T.L. (1966), Allocation of Resources - CPM, *Revue Française de Recherche Opérationelle*, 38, 31-38.
Patterson, J.H. (1976), Project Scheduling: The Effects of Problem Structure on Heuristic Performance, *Naval Research Logistics*, 23, 95-123.
Patterson, J. (1984), A Comparison of Exact Procedures for Solving the Multiple Constrained Resource Project Scheduling Problem, *Management Science*, 30, 854-867.
Patterson, J.H. and W.D. Huber (1974), A Horizon-Varying Zero-One Approach to Project Scheduling, *Management Science*, 20, 990-998.
Patterson, J.H. and G. Roth (1976), Scheduling A Project Under Multiple Resource Constraints: A Zero-One Programming Approach, *AIIE Transactions*, 8, 449-456.
Patterson, J.H., R. Slowinski, F.B. Talbot and J. Weglarz (1989), An Algorithm for A General Class of Precedence and Resource Constrained Scheduling problems, in Slowinski, R. and J. Weglarz (eds.), *Advances in Project Scheduling*, Elsevier, Amsterdam.
Patterson, J.H., R. Slowinski, F.B. Talbot and J. Weglarz (1990), Computational Experience with a Backtracking Algorithm for Solving a General Class of Precedence and Resource-Constrained Scheduling Problems, *European Journal of Operational Research*, 49, 68-79.
Petroviç, R. (1968), Optimisation of Resource Allocation in Project Planning, *Operations Research*, 16, 559-586.
Pinedo, M. (1995), *Scheduling - Theory, Algorithms and Systems*, Prentice-Hall, Englewood Cliffs, New Jersey.
Pinson, E. (1994), Mémoire d'habilitation à diriger des recherches, Université Pierre et Marie Curie (Paris VI).

Pritsker, A.B., L.J. Watters and P.M.Wolfe (1969), Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach, *Management Science*, 16, 93-108.

Radermacher, F.J. (1985), Scheduling of Project Networks, *Annals of Operations Research*, 4, 227-252.

Rinnooy Kan, A.H.G. (1976), *Machine Scheduling Problems - Classification, Complexity and Computations*, M. Nijhoff.

Schrage, L. (1970), Solving Resource-Constrained Network Problems by Implicit Enumeration - Nonpreemptive Case, *Operations Research*, 10, 263-278.

Schwindt, C. (1995), ProGen/max: A New Problem Generator for Different Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags, Technical Report WIOR-449, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.

Schwindt, C. (1996), private communication.

Schwindt, C. and K. Neumann (1996), A New Branch-and-Bound-Based Heuristic for Resource-Constrained Project Scheduling with Minimal and Maximal Time Lags, *PMS'96 - Fifth International Workshop on Project Management and Scheduling*, April 11-13, Poznan, 212-215.

Shmoys, D.B. and E. Tardos (1993), Computational Complexity of Combinatorial Problems, in Graham, R.L., M. Grötschel and L. Lovasz (eds.), *Handbook in Combinatorics*, North-Holland, Amsterdam.

Simpson III, W.P. and J.H. Patterson (1996), A Multiple-Tree Search Procedure for the Resource-Constrained Project Scheduling Problem, *European Journal of Operational Research*, 89, 525-542.

Slowinski, R. (1980), Two Approaches to problems of Resource Allocation Among Project Activities - A Comparative Study, *Journal of the Operational Research Society*, 31, 711-723.

Speranza, M.G. and C. Vercellis (1993), Hierarchical Models for Multi-Project Planning and Scheduling, *European Journal of Operational Research*, 64, 312-325.

Sprecher, A. (1994), Resource-Constrained Project Scheduling: Exact Methods for the Multi-Mode Case, Lecture Notes in Economics and Mathematical Systems, Vol. 409, Springer, Berlin.

Sprecher, A. and A. Drexl (1996a), Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. Part I: Theory, Research report 385, Christian-Albrechts-Universität zu Kiel, Germany.

Sprecher, A. and A. Drexl (1996b), Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General and Powerful Sequencing Algorithm. Part II: Computation, Research report 386, Christian-Albrechts-Universität zu Kiel, Germany.

Sprecher, A., S. Hartmann and A. Drexl (1994), Project Scheduling with Discrete Time-Resource and Resource-Resource Trade-Offs, Research report 357, Christian-Albrechts-Universität zu Kiel, Germany.

Stinson, J.P., E.W. Davis and B.M. Khumawala (1978), Multiple Resource-Constrained Scheduling Using Branch-and-Bound, *AIIE Transactions*, 10, 252-259.

Talbot, F.B. (1982), Resource-Constrained Project Scheduling with Time-Resource Trade-Offs: The Nonpreemptive Case, *Management Science*, 28, 1197-1210.

Talbot, B. and J.H. Patterson (1978), An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems, *Management Science*, 24, 1163-1174.

Tanaev, V.S., V.S. Gordon and Y.M. Shafransky (1994a), *Scheduling Theory: Single-Stage Systems*, Kluwer Academic Publ., Dordrecht.

Tanaev, V.S., Y.N. Sotskov and V.A. Strusevich (1994b), *Scheduling Theory: Multi-Stage Systems*, Kluwer Academic Publ., Dordrecht.

Weglarz, J. (1981), Project Scheduling with Continuously-Divisible, Doubly-Constrained Resources, *Management Science*, 27, 1040-1053.

Wiest, J.D. (1964), Some Properties of Schedules for Large Projects with Limited Resources, *Operations Research*, 12, 395-418.

Wikum, E.D., C.L. Donna and G.L. Nemhauser (1994), One-Machine Generalized Precedence Constrained Scheduling Problems, *Operations Research Letters*, 16, 87-99.

Yang, K.K., F.B. Talbot and J.H. Patterson (1992), Scheduling a Project to Maximize Its Net Present Value: An Integer Programming Approach, *European Journal of Operational Research*, 64, 188-198.

Zahn, J. (1994), Heuristics for Scheduling Resource-Constrained Projects in MPM Networks, *European Journal of Operational Research*, 76, 192-205.